UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

Advanced automatic analysis of Cloud Chamber images

Relatore

Prof. Marco Zanetti

Laureando

Giacomo Barzon

Anno Accademico 2017/2018

Abstract

Cloud chambers were among the first particle detectors ever conceived; a few Nobel prizes have been awarded in the first half of the past century to discoveries made by means of such device, most importantly the positron and the muon. A cloud chamber inspired by those original apparatuses but improved in several respects have been recently realized in our labs. The work proposed in the context of this thesis concerns with the analysis of the images gathered by the cloud chamber exposed to natural environmental radiation. In addition to the necessary tuning and optimization of the chamber and camera parameters, the primary goal is to set up an automatic particle identification technique, based on advanced machine learning classification algorithms. In the future unsupervised learning methods will be tested in order to categorize a few of the main particle track types automatically, i.e. without teaching the algorithm with labeled image examples.

Sommario

Le camere a nebbia furono tra i primi rivelatori di particelle mai ideati; nella prima metà del secolo scorso alcuni premi Nobel sono stati assegnati a scoperte fatte per mezzo di tali dispositivi, in particolare il positrone e il muone. Una camera a nebbia ispirata a quegli apparati originali ma ottimizzata sono diversi aspetti è stata recentemente realizzata nei nostri laboratori. Il lavoro proposto nel contesto di questa tesi riguarda l'analisi delle immagini raccolte dalla camera a nebbia esposta alla radiazione ambientale naturale. Oltre alla necessaria messa a punto e ottimizzazione della camera e delle impostazioni della videocamera, l'obiettivo principale è quello di testare una tecnica di identificazione automatica delle particelle, basata su algoritmi avanzati di apprendimento automatico per la classificazione. In futuro verranno testati metodi di apprendimento non-supervisionato al fine di classificare automaticamente alcune delle tracce dei principali tipi di particelle, cioè senza istruire l'algoritmo con esempi di immagini etichettate.

Indice

Introduzione

Nei primi anni del secolo scorso protone, neutrone e elettrone avevano assunto il ruolo di costituenti elementari della materia. La scoperta di nuove particelle subatomiche portò a rivedere questa concezione e ad indagare le forze fondamentali che ne governano le interazioni. Ciò fu possibile mediante l'analisi della radiazione proveniente dallo spazio che interagisce con la nostra atmosfera: sciami di particelle che vengono chiamati raggi cosmici. Nacque così la branca della fisica chiamata fisica delle particelle. Lo strumento che permise queste scoperte fu la camera a nebbia.
Ideato da Wilson nel 1911 mentre "giocava" con la nebbia studiando il fenomeno della condensazione, nella seconda metà del 900 venne via via soppiantato da strumenti sempre più efficienti e precisi nella corsa alla scoperta di tutto quello zoo di particelle la cui descrizione è stata unificata nel Modello Standard.
Ai giorni nostri la camera a nebbia mantiene comunque un utilizzo pedagogico vista la relativa semplicità per la sua realizzazione e l'efficacia con cui rende visibili ad occhio nudo le traiettorie delle particelle e ci permette di intuirne la presenza intorno a noi e le loro proprietà fisiche.

Questo strumento può essere anche utile per testare le più recenti tecniche di classificazione automatica dei vari tipi di particelle presenti nella radiazione ambientale, basate su algoritmi di apprendimento automatico. Questi algoritmi sono in grado di imparare dall'esperienza e acquisire le proprie conoscenze estraendo schemi dai dati grezzi. Tale approccio evita la necessità per gli operatori umani di specificare formalmente tutte le proprietà e le caratteristiche dei diversi oggetti della classificazione. I più moderni algoritmi di apprendimento automatico vengono utilizzati ad esempio per le Deep Neural Network: analizzando milioni, se non miliardi, di dati ottengono un'efficienza e precisione maggiore rispetto ad approcci classici in svariati campi, che vanno dal riconoscimento della scrittura manoscritta al controllo di qualità su produzioni industriali, dalla traduzione di un testo tra varie lingue alla diagnostica medica.

Lo scopo di questa tesi è per l'appunto l'applicazione di algoritmi di apprendimento automatico per il riconoscimento e l'identificazione delle particelle nelle immagini ottenute da una camera a nebbia, che è stata costruita nei laboratori del Dipartimento, esposta alla radiazione ambientale. In particolare, è stata utilizzata la struttura della Rete Neurale Convoluzionale, allenata con algoritmi di apprendimento supervisionato. L'obiettivo futuro sarà l'utilizzo del paradigma di apprendimento non-supervisionato per allenare la rete nell'identificazione delle particelle. In questo modo si vuole vedere come il sistema riclassificherà ed organizzerà una serie di input non appartenenti a classi predefinite sulla base di caratteristiche comuni.

Nella prima parte di questa tesi viene inizialmente descritto il funzionamento generale di una camera a nebbia (Capitolo 1) e le caratteristiche specifiche della camera del laboratorio (Capitolo 2). Nella seconda parte si descrive il processo di classificazione delle tracce generate dal passaggio delle particelle nella camera: dall'algoritmo per l'estrazione delle singole dalle immagini (Capitolo 3) alle caratteristiche di una rete neurale convoluzionale (Capitolo 4). Infine vengono presentati i risultati ottenuti con varie architetture della rete (Capitolo 5).

Parte I

La Camera a Nebbia

Capitolo 1

La fisica della Camera a Nebbia

La camera a nebbia è uno dei più datati e semplici rivelatori di particelle elementari. Il principio fondamentale del suo funzionamento è la sovrasaturazione di una sostanza gassosa, cioè uno stato "metastabile" in cui aria o un qualsiasi gas contiene più vapore di quanto ne contenga all'equilibrio stabile. Il passaggio di particelle cariche provoca la ionizzazione delle molecole del gas, che in questa particolare condizione si comportano come nuclei di condensazione. La condensazione del gas attorno a questi nuclei rende visibile la traiettoria delle particelle e permette uno studio delle loro proprietà fisiche. Nel corso degli anni, questa idea ha portato alla creazione di vari tipi di camera a nebbia sempre piu' sensibili e a vari utilizzi, ma a partire dagli anni 60 è stata soppiantata da rivelatori con maggiore risoluzione e efficienza, quali camere a bolle, calorimetri e scintillarori. Lo scopo di questo capitolo è mostrarne lo sviluppo e l'importanza storica che ha avuto nel corso del secolo passato (Sez.1.1) e i principi fisici che ne governano il funzionamento (Sez.1.2 e 1.3).

1.1 Storia

Le camere a nebbia vennero inizialmente studiate al Cavendish Laboratory da C.T.R. Wilson nel 1911 durante le sue ricerche sulla condensazione dei nuclei e la formazione della nebbia [1]. La condizione di sovrasaturazione era raggiunta solamente per pochi secondi raffreddando periodicamente l'aria sotto il punto di condensazione dell'acqua grazie ad un'espansione adiabatica. La struttura originale si può osservare in Figura 1.1.

Tale invenzione divenne di fondamentale importanza per lo sviluppo della fisica subnucleare nella prima metà del '900 a tal punto che Wilson nel 1927 venne insignito del Premio Nobel per la fisica: alcune delle scoperte più rilevanti furono la conferma dell'effetto Compton con la scoperta degli elettroni di recoil fatta dallo stesso Wilson, la scoperta della prima antiparticella, il postitrone (Figura 1.2(a)), la scoperta della prima particella della seconda generazione, il muone, e della prima particella strana, il kaone (Figura 1.2(b)).

Sebbene nel corso degli anni Wilson abbia migliorato il funzionamento della camera a nebbia, soffriva ancora di essere intrinsecamente intermittente nel suo utilizzo. Questo problema è stato parzialmente superato utilizzando dei contatori Geiger posti sopra e sotto la camera come trigger per azionare l'espansione al passaggio di una particella. Nel 1938 A.Langsdorf

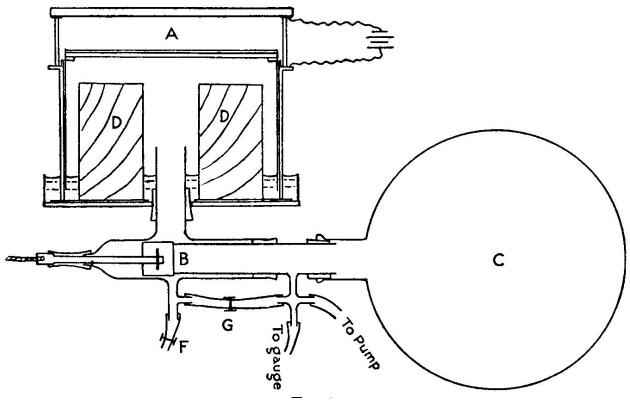
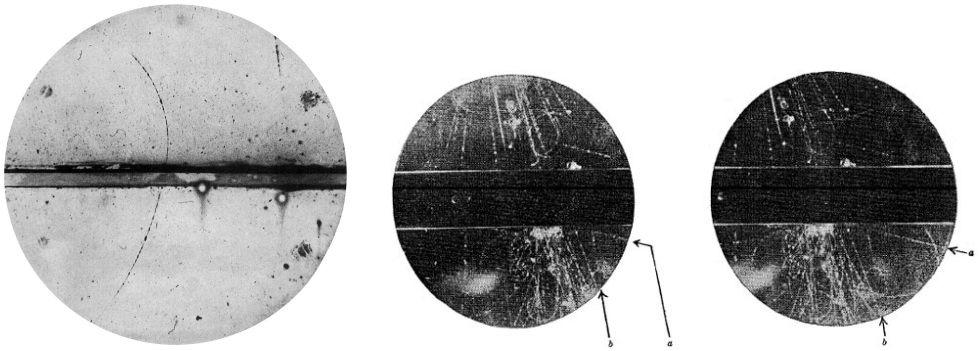


Figura 1.1: Struttura originale della camera a nebbia ad espansione adiabatica. Il meccanismo consisteva in un pistone B che periodicamente metteva in contatto la camera A con il contenitore di vuoto C o con l'ambiente esterno F, causando quindi un'espansione adiabatica [1]



(a) *Positrone* (b) *Kaone*

Figura 1.2: Fotografie di una camera a nebbia in cui sono visibili le tracce originali di positrone e kaone.

nella sua Ph.D. tesis presentò quella che chiamò *continuosly sensitive diffusion cloud chamber* (camera a nebbia a diffusione sensibile continua) [2]. Il suo progetto originario si può vedere in Figura 1.3.
Questo tipo di camera è costituito da un volume in cui il fondo viene raffreddato al di sotto della temperatura di condensazione della sostanza utilizzata. Questa viene riscaldata e il vapore viene lasciato diffondere (per nebulizzazione o evaporazione) dall'alto verso il basso attraverso un altro gas, tipicamente l'aria. Incontrando un gradiente di temperatura il gas condensa, producendo continuamente vapore sovrassaturo e quindi la nebbia: questa modalità permette di fotografare le tracce con più facilità.

Le camere a nebbia furono lo strumento principe per la rivelazione di particelle fino agli anni 50. La maggior parte delle ricerche per lo sviluppo e il perferzionamento delle camere a nebbia si fermò a partire dal 1952 quando D.A. Glaser inventò la camera a bolle. Le camere a bolle utilizzano liquido, solitamente idrogeno liquido, surriscaldato oltre il punto di ebollizione, tale da provocare nuclei di ionizzazione con anche piccole perturbazioni dovute a particelle cariche. L'utilizzo di un liquido al posto di un gas come origine di nuclei di condensazione ha il vantaggio di offrire maggiore densità, di conseguenza miglior potere di frenamento, maggiore ionizzazione specifica e quindi migliore definizione delle tracce e maggiore risoluzione spaziale. Queste camere sono state a loro volta sostituite dalle camere a multifili, che permettono invece

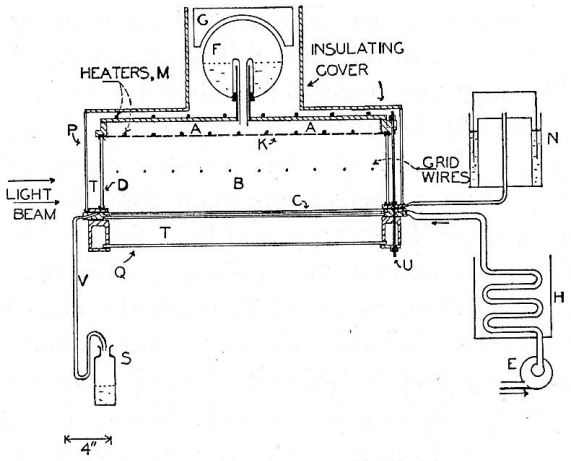


Figura 1.3: Idea originale della camera a diffusione di Langsdorf. S, N e H sono rispettivamente i contenitori di vapore condensato, gas e liquido per il raffreddamento. [2]

di rilevare elettronicamente le traiettorie delle particelle.

1.2 Condensazione

La condensazione (anche detta liquefazione) è la *transizione di fase dalla fase aeriforme alla fase liquida* di una sostanza. Il fenomeno della condensazione gioca un ruolo fondamentale nelle camere a nebbia. Proprio lo studio sul fenomeno della condensazione e della formazione della nebbia portò Wilson all'ideazione di questo strumento per la rivelazione di particelle. I blob di nebbia consistono infatti nel risultato finale del passaggio della particella.

Di seguito si analizzano le condizioni affinche' sia possibile la formazione di gocce. In un sistema termodinamico in sovrasaturazione la formazione delle gocce è un processo molto disomogeneo rispetto alla condizione di saturazione. Infatti, come inizialmente mostrato da Lord Kelvin e Robert von Helmholtz, a causa della tensione superficiale la pressione di vapore saturo di una goccia di raggio r, p_r, è più grande della pressione di vapore saturo su di una superficie piana, p_∞. La pressione di vapore di una determinata sostanza è la pressione esercitata dal vapore sulla fase condensata (solida o liquida) della medesima sostanza quando tali fasi sono in condizioni di vapore saturo, cioè in condizioni di equilibrio termodinamico tra di loro all'interno di un sistema chiuso. L'equazione di Kelvin o di Ostwald-Freundlich descrive l'andamento della pressione di vapore su di un'interfaccia curva tra liquido e vapore, come nel caso di una goccia:

$$p_r = p_\infty exp\left(\frac{r_{crit}}{r}\right) = p_\infty exp\left(\frac{2\gamma}{r}\frac{M}{RT\rho}\right) \tag{1.1}$$

ove:

- γ è la tensione superficiale
- ρ è la densita' del liquido che si e' formato
- M è la sua massa molare

- R è la costante universale dei gas
- T è la temperatura del sistema
- r_{crit} è il raggio critico, ossia la dimensione minima oltre la quale avviene la formazione della goccia

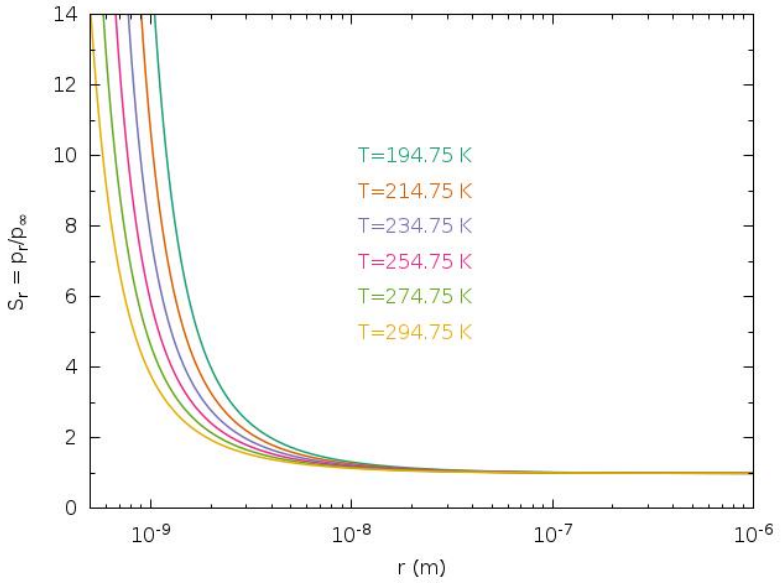


Figura 1.4: Pressione di vapore per gocce sferiche di alcool isopropilico a differenti temperature [3].

Dato un sistema con una certa pressione p, spesso ci si riferisce alla quantità adimensionale $\frac{p}{p_\infty}$ come *sovrasaturazione* S. Si può facilmente osservare dalla formula 1.1 come nel caso in cui $S < 1$ le goccioline siano instabili, in quanto la pressione p_r di una goccia è sempre maggiore della pressione di saturazione p_∞: necessitano dunque di una pressione sulla superficie maggiore della pressione di saturazione p_∞ e una qualsiasi gocciolina creatasi evapora. Inoltre, tale equazione mostra che le gocce non si possono creare spontaneamente, visto che per una goccia di raggio infinitesimo è richiesto un valore di sovrasaturazione esponenzialmente alto. Se invece viene formata una goccia di raggio abbastanza grande, la sovrasaturazione puo' essere raggiunta e la goccia è stabile. Inoltre, quando il vapore condensa su di una goccia gia' formata, il raggio di questa aumenta; come si puo' vedere dalla Figura 1.4 la pressione di condensazione p_r richiesta diminuisce e la goccia continua a crescere fino a raggiungere una grandezza visibile.

I *nuclei di condensazione* sono microparticelle di varia natura fortemente igroscopiche che favoriscono sensibilmente la condensazione su di esse, dando origine al processo e alla formazione delle gocce. In assenza dei nuclei il raggiungimento dei cosiddetti raggi critici richiede gradi di sovrassaturazione troppo elevati.

Tale fenomenologia cambia in maniera rilevante se la goccia è carica elettricamente, che e' proprio il caso di funzionamento delle camere a nebbia, visto che le gocce si formano attorno alle molecole ionizzate dal passaggio delle particelle subnucleari.

La formula 1.1 diviene quindi:

$$p_r = p_\infty exp\left[\frac{M}{RT\rho}\left(\frac{2\gamma}{r} - \frac{e^2}{32\pi^2\epsilon r^4}\right)\right] \tag{1.2}$$

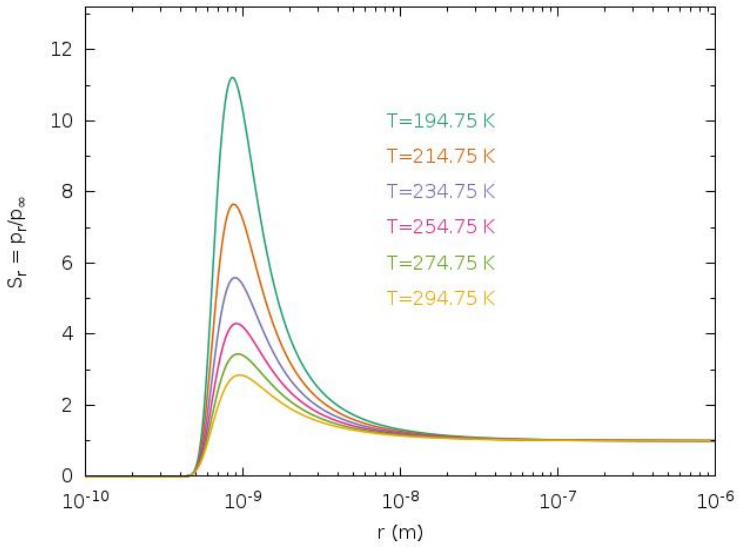


Figura 1.5: Pressione di vapore per gocce sferiche cariche di alcool isopropilico a differenti temperature [3].

ove ϵ e' la costante dielettrica del mezzo circostante.

Come si può osservare dalla Figura 1.5 esiste un valore massimo di sovrasaturazione che separa due regioni fenomenologicamente differenti; una prima, in cui S_r è una funzione decrescente del raggio e di conseguenza le gocce non crescono spontaneamente, ma a differenza del caso neutro per nuclei carichi infinitesimamente piccoli le gocce si possono formare. Una seconda regione in cui invece S_r è decrescente rispetto al raggio. Per un dato valore di S sopra un certo limite esistono quindi due raggi notevoli $r_c(S)$ per cui $p_{r_c}/p_\infty = S$: per ogni $r < r_{c,min}$ le gocce crescono fino a raggiungere una dimensione stabile $r_{c,min}$, mentre ogni goccia di raggio $r > r_{c,max}$ puo' crescere spontaneamente fino a raggiungere una dimensione visibile. La caratteristica importante è che se la sovrasaturazione del sistema è abbastanza elevata, il raggio $r_{c,min}$ delle gocce che si possono formare spontaneamente puo' raggiungere il massimo della Figura 1.5 e quindi crescere spontaneamente: cio' comporta che ogni singola particella carica si comporta come un nucleo di condensazione e di conseguenza ogni particella ionizzante che attraversa la camera genera un numero sufficiente di ioni da lasciare una traccia visibile.

1.3 Interazione delle particelle con la materia

Nella Sezione 1.2 si è osservato come il processo che da inizio alla nucleazione sia la ionizzazione del gas da parte delle particelle cariche che attraversano la camera. Le particelle elettricamente cariche interagiscono principalmente per via elettromagnetica con le molecole del gas e provocano la rottura dei legami molecolari e atomici, liberando coppie di ioni. A partire dalla formula di Bethe-Bloch si puo' stimare la *ionizzazione specifica* dN/dx, definita come il numero medio di coppie di ioni prodotte per unita' di lunghezza in un determinato mezzo dovuto al passaggio di una particella massiva:

$$\frac{dN}{dx} = \frac{\rho}{W} K z^2 \frac{Z}{A} \frac{1}{\beta^2} \left[\frac{1}{2} \ln \left(\frac{2 m_e c^2 \beta^2 \gamma^2 W}{I^2} \right) - \beta^2 - \frac{\delta}{2} \right] \tag{1.3}$$

ove:

- ρ è la densità del materiale assorbitore (nel caso in esame il gas utilizzato)
- $K = 4\pi N_A r_e^2 m_e c^2 = 0.307075\ MeV cm^2 g^{-1}$, ove m_e è la massa dell'elettrone, r_e è il raggio classico dell'elettrone, N_A è la costante di Avogadro
- z è la carica della particella incidente
- $\frac{Z}{A}$ è il rapporto tra il numero atomico e il numero di massa del materiale assorbitore
- β è la velocità della particella incidente, γ il corrispondente fattore di Lorentz
- I è il potenziale medio di ionizzazione del mezzo assorbitore
- W è l'energia media necessaria per la produzione di una coppia ione-elettrone. Questa grandezza dipende in modo trascurabile dall'energia della particella incidente e vale circa 25-40 eV
- δ è il fattore dell'effetto densità, un termine correttivo che tiene conto della polarizzazione del mezzo assorbitore indotta dal campo elettrico della particella che lo attraversa. E' un fattore che diventa rilevante ad alte energie.

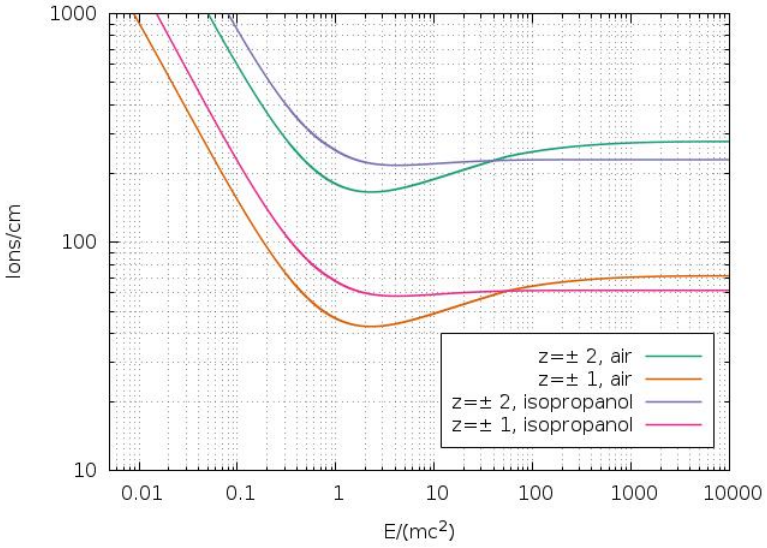


Figura 1.6: Ionizzazione specifica per differenti valori di carica e mezzi assorbenti [3].

La formula 1.3 deve tenere conto di alcune correzioni nel caso in cui le particelle incidenti abbiano massa comparabile a quella dell'elettrone, come nel caso di positroni o gli stessi elettroni : ciò è causato dall'effetto di Bremsstrahlung, ossia la perdita di energia per emissione di radiazione dovuta all'accelerazione o decelerazione della particella in un campo elettromagnetico, fenomeno che diventa più rilevante per urti ad alte energie o interazione tra particelle di massa comparabile.

Se si considera la formula 1.3 in funzione di $\beta\gamma = \frac{p}{mc}$ (ove le varie grandezze che si riferiscono alla particella incidente), la ionizzazione specifica assume un andamento asintotico che dipende da $\frac{1}{\beta^2}$ per valori di $\beta\gamma < 1$, mostra un minimo per $\beta\gamma \approx 3,5$, detto minimo di ionizzazione, e successivamente risale come $\ln(\beta\gamma)^2$, saturando poi per gli effetti di densità. Inoltre, si può osservare dalla formula 1.3 che è dipendente dalla densità del mezzo assorbitore; questa è la motivazione per cui le camera a nebbia negli anni sono state sostituite da altri tipi di

rivelatori, che come mezzo interagente utilizzano un liquido, come nella camera a bolle, o un materiale solido, come i calorimetri: infatti vista la maggiore densità favoriscono la produzione di ioni e rendono più semplice e precisa la rivelazione di particelle subatomiche. Si può anche osservare come la ionizzazione prodotta sia indipendente dalla massa della particella incidente, ma dipenda unicamente dall'energia che questa possiede.

Nel procedere all'interno di un mezzo materiale, le particelle cariche perdono quindi progressivamente energia e rallentano; se il materiale attraversato è abbasanza spesso, queste possono arrivare anche al completo frenamento al suo interno. Lo spazio percorso dalla particella fino al frenamento si definisce *range*. Questa distanza si puo così stimare:

$$R = \int_E^0 \left(\frac{dE}{dx}\right)^{-1} dE \tag{1.4}$$

Le particelle cariche sono generate da processi radioattivi, come i decadimenti (di tipo α o β) di nuclei instabili, o processi collisionali tra particelle ad alta energia. Tali tipi di fenomeni possono avere origine naturale, ad esempio decadimenti di radioisotopi primordiali presenti nell'ambiente, tipicamente potassio-40 o le famiglie radioattive di Uranio e Torio (che corrispondono a circa 80% di tutta la radiazione ambientale); altre particelle cariche presenti in natura sono i residui di sciami cosmici prodotti da particelle extraterrestri che interagiscono con l'atmosfera e danno origine a varie catene di decadimento (circa il 15%); possono avere anche origine da collisioni controllate, come avviene negli acceleratori di particelle.

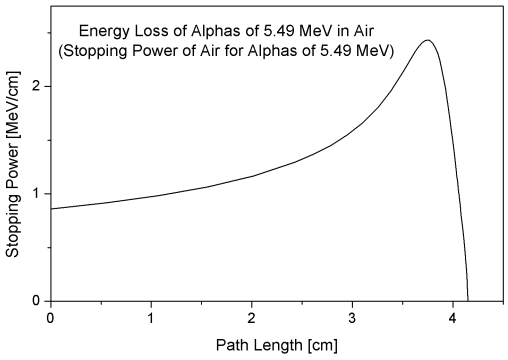


Figura 1.7: Curva di Bragg per particelle alfa in aria. L'intercetta della curva è una stima del range della particella.

La camera a nebbia puo essere utilizzata per rivelare la radiazione ambientale dell'ambiente circostante o per studiare il decadimento di un certo tipo di materiale radioattivo, inserendolo direttamente all'interno della camera.

Capitolo 2

L'apparato sperimentale

Nei laboratori del Dipartimento di Fisica è stata costruita una camera a nebbia a diffusione funzionante con alcool isopropilico (2-propanolo). La camera è stata equipaggiata di un sistema di ripresa per poter analizzare le immagini delle tracce create della radiazione ambientale. In questo capitolo vengono descritte le caratteristiche principali della camera, in particolare le sue componenti strutturali (Sez. 2.1) e gli elementi di elettronica e software del sistema di controllo (Sez. 2.2). Sono state cercate le condizioni sperimentali ottimali per la presa dati, testando vari setup per la temperatura, l'illuminazione e le impostazioni di ripresa della videocamera (Sez. 2.3).

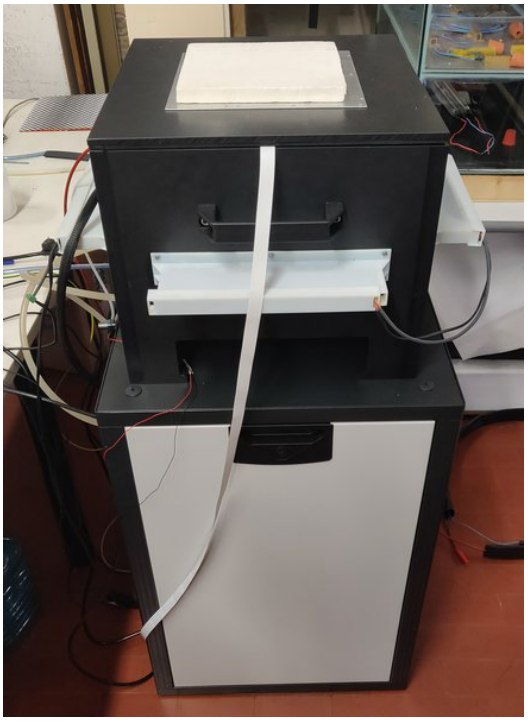


Figura 2.1: Camera a nebbia del Dipartimento. In alto, è alloggiata la camera sopra di una struttura per contiene le componenti elettroniche, i vari generatori di corrente e il chiller.

2.1 Materiali e strumentazione

Struttura. La struttura principale della camera è una cassa in pvc di dimensioni 40cm x 40cm x 15cm (se si comprendono anche gli appoggi inferiori 40cm x 40cm x 35cm). La faccia superiore è mancante per permettere l'alloggiamento di un vetro riscaldabile, così che si possa osservare ad occhio nudo le tracce che si formano all'interno della camera. Al centro di ciascuna delle facce laterali è presente una fessura di 1 centimetro di spessore per l'alloggiamento del sistema di illuminazione, che consiste in una doppia fila di led di luce bianca (Figura 2.2). All'interno della cassa è situata una struttura metallica come mostrato in Figura 2.3, costituita di una canaletta in cui alloggia l'alcol liquido rialzata di larghezza 3,5cm e profondità 2cm che scorre lungo il perimetro. All'interno della struttura sono stati saldati longitudinalmente 8 fili paralleli tra loro per generare il campo elettrico. Sul fondo è stato posizionato invece un piatto di alluminio collegato invece al sistema di raffreddamento e al suo intorno è stato creato un canale di scolo per l'alcol. Al di sotto della cassa è stato posizionato uno strato di polistirolo che funge da strato isolante e un piccolo serbatoio per il recupero del liquido e per la sua immissione nella canaletta tramite una pompetta elettrica.

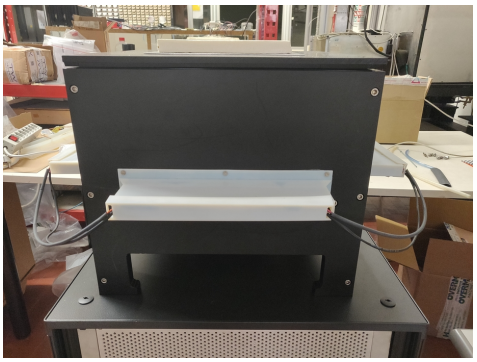


Figura 2.2: Vista laterale della camera. La struttura bianca è uno dei quattro alloggiamenti per le strisce di led, che comunicano direttamente con l'interno.

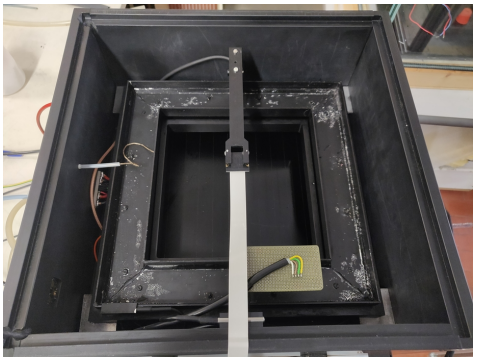


Figura 2.3: Vista superiore della camera senza copertura. Si vede la canaletta per l'alloggiamento dell'alcool liquido, il supporto per la telecamera al centro e sulla sinistra il tubo della pompetta.

Sistema di riscaldamento. Per riscaldare tutta la zona superiore della camera e permettere all'alcol liquido di raggiungere il punto di evaporazione, si utilizzano una serie di resistenze poste lungo tutto il percorso della canaletta. Questo sistema riscalda la canaletta fino a quasi $50°C$.

Sistema di raffreddamento. La camera a condensazione necessita però di una zona con grande differenza di temperatura per poter funzionare. Il solo riscaldamento della regione superiore non permette di raggiungere le condizioni di sovrasaturazione. Per aumentare il gradiente verticale di temperatura viene utilizzato un chiller con raffreddamento ad acqua, costituito da un sistema di *celle di Peltier* ceramiche situate al di sotto del piatto di alluminio che raffreddano la parte inferiore della camera fino a circa $-10°C$.

Campo elettrico. Per aumentare l'efficienza della camera e il numero di particelle che entrano nella regione sensibile, è stata dotata la camera di un campo elettrico verticale.

Per fare ciò, come spiegato in precedenza sono stati saldati dei fili di materiale conduttore alla struttura della canaletta, i quali vengono collegati ad un polo di un generatore di alta tensione; il piatto di alluminio viene invece collegato al polo negativo, creando così una differenza di potenziale verticale.

Tutti gli elementi interni alla camera sono stati ricoperti con una vernice nera resistente all'alcol per aumentare il contrasto delle tracce rispetto al fondo e diminuire il più possibile i rilfessi dell'illuminazione.

2.2 Sistema di controllo

Il controllo di tutta la camera, l'acquisizione e la gestione dei dati viene comandato da un Raspberry Pi. A questo sono stati collegati vari sensori, una pompetta e una videocamera. E' stato scritto un software che gestisce in automatico il funzionamento della camera, come ad esempio il refill automatico di alcool nella canaletta.

Sensori. Vengono utilizzati tre sensori di temperatura: un sensore monitora la temperatura del fondo, un secondo quella della canaletta e un altro quella del vetro superiore. In questo modo si è in grado di tracciare il valore massimo e minimo di temperatura all'interno della camera. E' stato installato anche un sensore di distanza a ultrasuoni che monitora costantemente l'altezza del liquido nella canaletta e permette di attivare in automatico la pompetta. In questo modo si ha una presenza costante di alcool nella camera per tutta la durata del suo funzionamento.

Videocamera. Gli eventi fisici che si producono in una camera a nebbia sono le tracce: per raccogliere gli eventi si utilizzia una videocamera (PiCamera[1]) che filma la zona attiva della camera. Da questi video si possono estrapolare i frame e analizzare le tracce che si sono generate: la videocamera è quindi il vero e proprio strumento di misura per la nostra analisi.

2.3 Punto di lavoro

Si sono cercate le condizioni sperimentali ottimali per ottenere il maggior grado di saturazione possibile e di conseguenza una migliore risoluzione delle tracce. I parametri che abbiamo controllato sono state le varie temperature e il valore di tensione da erogare per la generazione del campo elettrico. In Figura 2.4 e 2.5 si può osservare l'andamento della temperatura rispettivamente della canaletta e del fondo, misurati a partire dall'accensione della camera in cui tutte le componenti si trovano a temperatura ambiente.

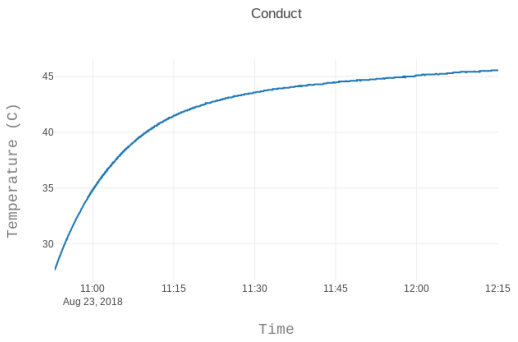


Figura 2.4: Andamento temporale della temperatura della canaletta.

[1]per le specifiche hardware si veda il datasheet: `https://www.raspberrypi.org/documentation/hardware/camera/`

La formazione della nebbia si inizia a osservare dopo circa 30 minuti, quando la canaletta si trova a circa $42°C$ e il fondo a circa $-5°C$. Il sistema raggiunge il punto di lavoro ottimale dopo circa 45 minuti ad una temperatura della canaletta di $45°C$ e $-9°C$ del fondo.

E' stato poi cercato il valore migliore di tensione che genera il campo elettrico nella camera. All'iniziale aumento della tensione il numero di tracce che si possono osservare aumenta di molto. Se si supera un certo valore, si possono generare delle scariche elettriche tra i fili. Queste scariche producono dei grandi ammassi di condensato.

Abbiamo quindi impostato la tensione al valore più grande possibile tale da non incorrere in questa problematica.
Fissate le condizioni migliori per la generazione delle tracce, in seguito abbiamo cercato il modo migliore per poterle osservare e registrare. In prima battuta abbiamo provato varie combinazioni di illuminazione, variando da quali lati illuminare la camera (solo lato anteriore e posteriore, solo lati laterali, tutti e quattro i lati; ciascuna combinazione è stata provata sia con una o entrambe le strisce di led accesi), combinandola alle impostazioni di ripresa della videocamera. Il setup migliore è risultato illuminare l'interno della camera da tutti e quattro i lati esterni con entrambe le strisce di led (quindi la condizione di luce massima ottenibile). La videocamera invece è stata così impostata[2] per aumentare il contrasto delle tracce rispetto al fondo:

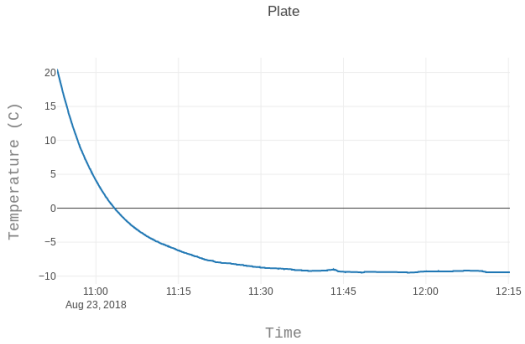


Figura 2.5: Andamento temporale della temperatura del fondo.

- ISO: 600
- Fps (frames per second): 25
  Ogni frame corrisponde ad un tempo di acquisizione pari a $\frac{1}{25}s = 0,04s$
- Risoluzione: 1920x1080
- Contrasto: 100
- Brightness: 80

[2] si veda documentazione PiCamera: `https://www.raspberrypi.org/documentation/raspbian/applications/camera.md`

Parte II

Analisi automatica delle immagini

Capitolo 3

Image preprocessing

Una parte fondamentale del processo di analisi consiste nella preparazione degli esempi per il training della rete. Nel caso specifico si tratta di un'immagine contenente una singola traccia lasciata da una particella al suo passaggio nella camera. Per fare ciò è stato scritto un algoritmo che automaticamente estrapola le tracce dai file video, applicando vari filtri di *denoising* alle immagini grezze per eliminare il fondo dovuto al moto caotico della nebbia (Sez.3.2.1) e algoritmi per la ricerca delle *componenti connesse* o *labeling* (Sez.3.2.2)[1]. Un fattore che è stato tenuto fortemente in considerazione nella scelta deli vari algoritmi e dei vari filtri è stata la necessità di eliminare dalle immagini i fili del campo elettrico, la cui presenza è una grande fonte di rumore sistematico.

3.1 Segmentazione delle immagini

In numerose applicazioni si presenta il problema di individuare gli oggetti distinti presenti in un'immagine o di partizionare un'immagine in regioni significative, operazione preliminare spesso necessaria nell'analisi e nell'interpretazione di immagini. Questo processo viene chiamato *segmentazione* dell'immagine. In ambito scientifico, ne trova molto uso in ambito medico per la diagnostica. Nel nostro caso, gli oggetti di interesse sono le tracce create dal passaggio di una particella nella camera a nebbia. Questi oggetti costituiscono quella che viene definita una *componente connessa* dell'immagine.

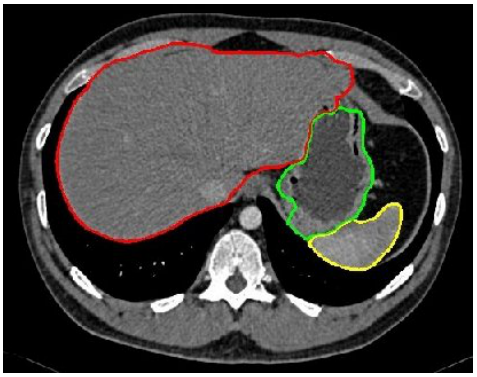


Figura 3.1: Segmentazione di un'immagine di una TAC addominale. L'algoritmo ha riconosciuto tre regioni: fegato (rosso), stomaco (verde) e milza (gialla).

Per *componente connessa* si intende *un sottoinsieme S di un'immagine formato da pixel connessi tra loro*, cioè per i quali esiste un *path* composto da pixel appartenente al sottoinseme S. Definita una relazione di k-adiacenza, un *percorso* o *path* di lunghezza n tra due pixel,

[1] Il codice sorgente è disponibile al seguente link: `https://github.com/GiacomoBarzon/TracesAnalysis`

che chiamiamo P e Q, è una sequenza di $n+1$ pixel $P = P_0, P_1, P_2, ...P_N = Q$ tale che P_i è k-adiacente a P_{i-1} $\forall i = 1, ..N$. L'unione di tutte le componenti connesse di un'immagine viene detta *primo piano* o *foreground* dell'immagine; il complementare, cioè l'insieme di pixel dell'immagine che non appartengono a nessuna regione, viene invece definito *sfondo* o *background*.

> "The segmentation of the image(s) presented to an image analysis system is critically dependent on the scene to be sensed, the imaging geometry, configuration and sensor used to transduce the scene into a digital image, and ultimately the desired output (goal) of the system." A.Jain

Generalizzando queste considerazioni, le tecniche di ricerca e di raggruppamento di elementi omogenei in un insieme di dati, tali che gli oggetti appartenenti a un gruppo siano "simili" tra loro e differenti dagli oggetti negli altri gruppi, si dicono *clustering*. Le nozioni di similarità, prossimità, appartenenza ad una medesima regione, sono spesso concepite in termini di distanza in uno spazio metrico[2] multidimensionale.

3.2 Algoritmo per l'estrapolazione delle tracce

E' stato scritto un algoritmo in PYTHON che automatizza il processo di estrazione delle tracce. Per prima cosa fa un *denoising* delle immagini per aumentare il rapporto segnale-rumore delle tracce, utilizzando vari package per l'elaborazione di video e immagini, quali OPENCV[3], FFMPEG[4], PILLOW[5]. Sono state utilizzate anche alcune funzioni presenti nella libreria KERAS, in particolare quelle utili per il pooling. Infine andiamo a selezionare le singole tracce mediante algoritmi di labeling.

3.2.1 Denoising delle immagini

Al fine di ottenere la massima rimozione della presenza fili per il campo elettrico sono state provate varie metodologie: ad esempio, abbiamo inizialmente provato a eliminare delle colonne corrispondenti ai fili verticali, visto la loro staticita' in tutte le immagini; oppure abbiamo sottratto un'immagine di fondo, presa a camera spenta. Tali tentativi si sono rivelati pero' inefficienti, perchè lasciavano comunque dei pixel bianchi attorno alla zona limitrofa ai fili.
Il meccanismo che si e' rivelato essere il migliore consiste nel: analizzare singolarmente parti di video di 15 secondi da cui estrarre i frames; per ciascun frame da cui estrarre le tracce, si

[2]Spazio metrico: struttura matematica costituita da una coppia (𝕏,d) ove 𝕏 è un insieme e d una funzione

$$d : \mathbb{X}x\mathbb{X} \to \mathbb{R}$$

detta *distanza* o *metrica*, tale che per ogni $x, y, z \in \mathbb{X}$ si abbia:

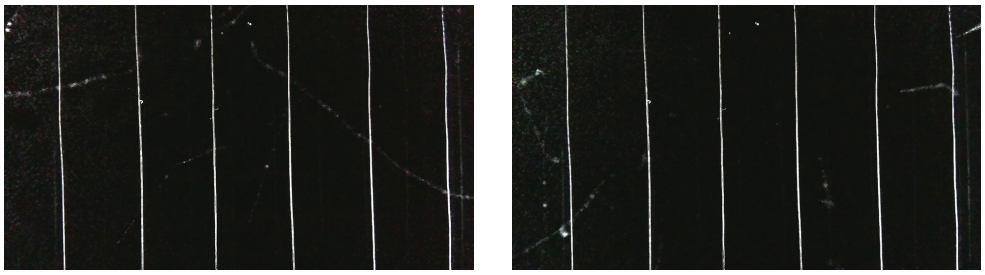
- $d(x, y) \geq 0$ e $d(x, y) = 0$ se e solo se $x = y$
- $d(x, y) = d(y, x)$
- $d(x, z) \leq d(x, y) + d(y, z)$

[3]Documentazione di OPENCV: https://opencv.org
[4]Documentazione di FFMPEG: https://www.ffmpeg.org
[5]Documentazione di PILLOW: https://pillow.readthedocs.io/en/5.2.x/

crea una serie di immagini di fondo prendendo altri frames dello stesso video, che si sottraggono uno ad uno al frame considerato; infine si media sulle varie differenze.

I principali passaggi del codice sono quindi stati:

- Suddivisione in campioni video di 15 secondi e estrazione dei frames:
  Sono stati estratti tramite le funzioni della libreria FFmpeg, selezionando un frame ogni cinque del video originale e sfruttando la funzione di *blending* di addizione, che consiste nel sommare il contributo di cinque frames consecutivi: in questo modo si aiuta l'estrazione delle tracce, dato che una traccia nella camera dura piu di cinque frames e risulta piu visibile nel frame cosi' estratto. I frame dell'analisi corrispondono quindi ad un tempo di acquisizione della videocamera pari a $0,04s \ x \ 5 = 0,2s$ (come già mostrato nella Sezione 2.3 ciascun frame catturato dalla videocamera corrisponde a 0,04s). Le immagini sono di dimensioni $1920x1080$ a toni di grigio[6].



(a) *Frame 10* (b) *Frame 20*

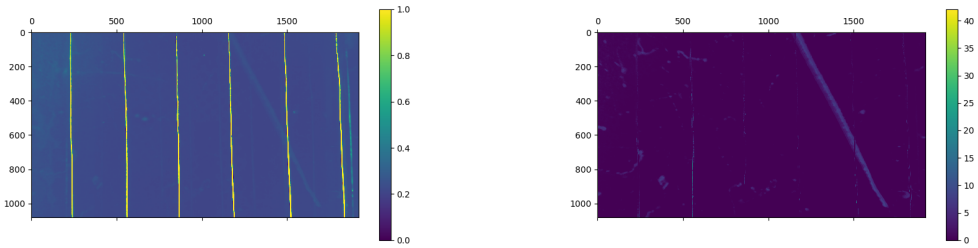
Figura 3.2: Due frame estratti dallo stesso campione video.

  In Figura 3.3 si possono invece osservare le matrici con rispettivamente media (normalizzata tra 0 e 1) e varianza della luminosità dei valori dei singoli pixel tra i frame di uno stesso campione. La media assume i valori maggiori in corrispondenza dei fili, che nonostante siano stati coperti di vernice scura riflettono comunque molto la luce all'interno della camera. Si notano anche altre linee verticali di minore luminosità, che corrispondono al riflesso dei fili sul piatto di alluminio situato sul fondo della camera. In questo video è stata catturata la traccia di una particella alfa, che è visibile sia sulla matrice media che quella della varianza.

- Trigger per la selezione dei frames di interesse:
  Per velocizzare l'algoritmo, è stato introdotto un trigger che seleziona i frame più interessanti, mentre esclude dall'elaborazione i rimanenti. Infatti è stato osservato che i frame in cui sono presenti degli eventi hanno valori di luminosità che si discostano maggiormente dalla media rispetto a frame privi di tracce. Definito $p_i[r][c]$ il valore (tra 0 e 255) che può assumere il pixel alla riga r e colonna c, è stato introdotto per ciascun frame il parametro

$$\mu_i = \frac{R_i}{\sigma} \tag{3.1}$$

[6]Per vedere un esempio di video della camera si veda il seguente link: `https://drive.google.com/file/d/1WNfxYRFqNKUfNrtVxth3LWb1qqk3BgBG/view?usp=sharing`



(a) *Media (normalizzata)* (b) *Varianza*

Figura 3.3: Media e varianza tra i valori di uno stesso pixel dei frames appartenenti ad un singolo campione video.

ove:

$$R_i = \sum_{row} \sum_{column} (p_i[r][c] - mean[r][c])^2 \tag{3.2}$$

è la somma dei quadrati della differenza tra i valori dei singoli pixel $p_i[r][c]$ del frame considerato e il valore medio in tutto il campione video di quel determinato pixel, mentre

$$\sigma = \sum_{row} \sum_{column} \sigma[r][c] \tag{3.3}$$

è la somma di tutti i pixel della matrice delle varianze $\sigma[r][c]$. I frame di interesse hanno tipicamente un valore di μ superiore a 0,004.

- Denoising iniziale e creazione immagini di background:
  Come immagini di fondo vengono selezionati un frame ogni quattro; a questi si applica: *non-local means denoising* [7]; un filtro uniforme[8], che consiste in una media locale dei valori dei pixel; *dilation* [9], che allarga gradualmente i confini delle regioni di pixel in primo piano. Quest'ultimo è efficace nella rimozione del bagliore prodotto dai fili del campo elettrico, perchè allarga la regione che andra' poi sottratta al frame analizzato.

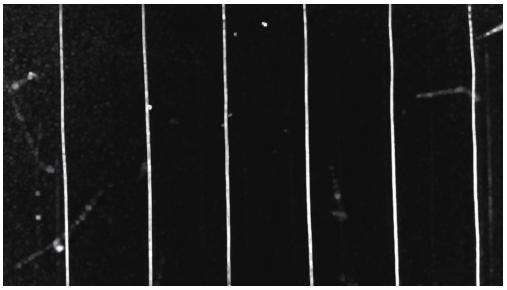
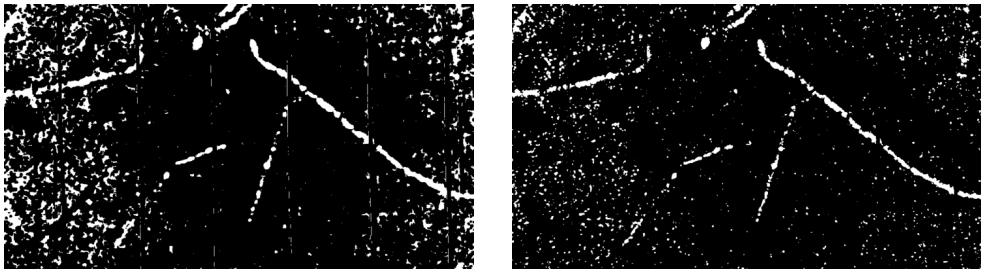


Figura 3.4: Esempio denoising frame di fondo (*frame 20*)

- Sottrazione del fondo, media e denoising finale:
  Al frame in analisi si sottraggono una ad una le immagini di fondo così analizzate; queste vengono poi binarizzate, ponendo un certo valore di soglia; in seguito si applica una funzione di *max-pooling*[10], che comporta una riduzione della dimensione dell'immagine

[7] Si veda `https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html`
[8] Si veda `http://www.mif.vu.lt/atpazinimas/dip/FIP/fip-Smoothin.html`
[9] Si veda `https://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm`
[10] Si veda `https://computersciencewiki.org/index.php/Max-pooling_/_Pooling`

a $960x540$; si media quindi tra tutti i frame cosi' creati e infine si applica nuovamente un filtro di binarizzazione.



(a) *Pooling e media frame sottratti* (b) *Frame finale*

Figura 3.5: Esempio media e denoising finale

3.2.2 Ricerca delle componenti connesse

Dopo aver rimosso il più possibile il fondo, l'algoritmo va a cercare le componenti connesse (si veda Sez.3.1). In letteratura e nei vari package sono presenti una vasta gamma di algoritmi di labeling e di clustering, che utilizzano differenti nozioni di distanza per definire i singoli raggruppamenti e di conseguenza ottengono risultati diversi. Per il nostro obiettivo è stato cercato l'algoritmo che meglio funzionasse nell'estrazione di oggetti allungati e che non riconducesse una singola traccia a due componenti separate; allo stesso tempo che non prendesse nell'oggetto troppi cluster di nebbia, rimanenti anche dopo il processo di denoising. Sono stati provati vari algoritmi di clustering; ma quello piu efficace si è dimostrato essere l'algoritmo di labeling del pacchetto NdImage [11].

Le tracce vengono poi centrate nell'immagine e ruotate in modo che la loro lunghezza maggiore risulti orizzontale. In più si scartano i bordi formati da pixel neri, riducendo la dimensione dell'immagine a 540x540. Infine le immagini sono state riscalate ad una dimensione di 128x128. Così facendo si diminuisce il numero di parametri di input e di conseguenza si alleggerisce il peso computazionale della rete, senza perdere comunque dettagli delle tracce. In aggiunta, l'operazione di rotazione elimina un grado di libertà nell'analisi delle tracce e semplifica il lavoro di classificazione della rete.

[11] Si veda `https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.label.html`

3.3 Esempi di tracce

Particelle beta. Le particelle più semplici da rivelare con una camera a nebbia sono le particelle beta. Queste particelle vengono espulse dai nuclei atomici per un processo chiamato decadimento beta. Esistono due forme di decadimento beta, uno denominato $\beta+$ in cui si ha produzione di positroni, un altro detto $\beta-$ in cui si ha invece produzione di elettroni.

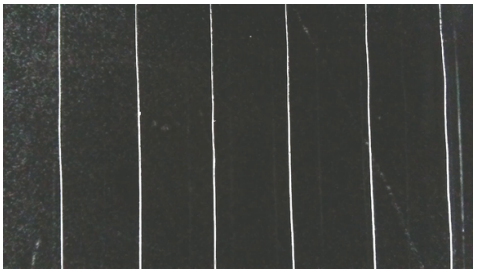


Figura 3.6: Frame 20 - esempio di traccia di una particella beta.

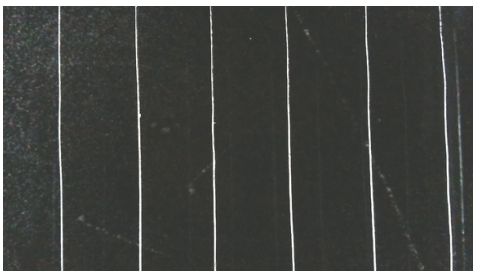
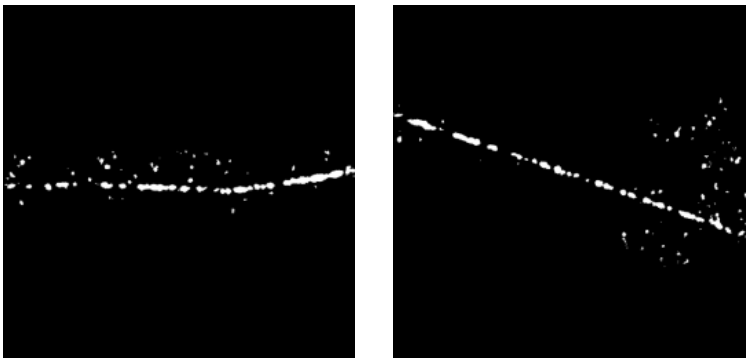


Figura 3.7: Frame 21 - esempio di traccia di una particella beta.

Tipicamente le tracce prodotte sono sottili e disomogenee. Questi tipi di particelle hanno una gamma di energie molto varia. A seconda dell'energia, si hanno range differenti e si possono osservare tracce lunghe come tracce più brevi. Nella Figura 3.6 e 3.7 si può vedere l'andamento temporale di una traccia beta sulla destra delle immagini. La produzione della traccia inizia a vedersi nel frame 19 e rimane visibile per i due frame successivi, che corrisponde ad un tempo di 0,4s. Per questo tipo di tracce il tempo che impiegano le gocce a generarsi è più lento di quanto impiegano poi a disperdersi.



(a) (b)

Figura 3.8: Tracce di particelle beta a fine elaborazione.

Particelle alfa. Un'altra particella molto comune in natura e facile da rivelare con la camera a nebbia è la particella alfa. Si tratta di un nucleo di ^4He, quindi è una particella massiva che interagisce fortemente con la materia vista la sua maggiore carica elettrica (z=2), come si può vedere dalla larghezza delle tracce prodotte. Infatti la traccia prodotta da una particella alfa è molto più larga e definita rispetto ad una traccia prodotta da una particella beta. In Figura 3.9, 3.10 e sono mostrati i frame contenenti un evento generato dal passaggio di una particella alfa. frame 40 il primo in cui si vede, 43 luminosità massima, dal 44 inizia a scendere come si vede nel 45, nel 46 quasi completamente scomparsa. Nel 39 non si vede niente, quindi la formazione della traccia è molto rapida, molto meno a scomparire.

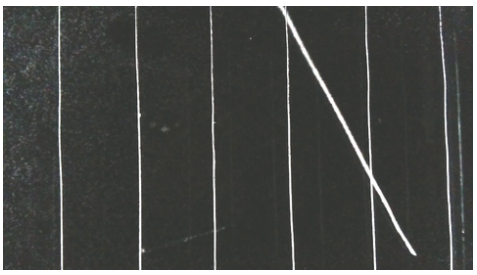


Figura 3.9: Frame 40 - esempio di traccia di una particella alfa.

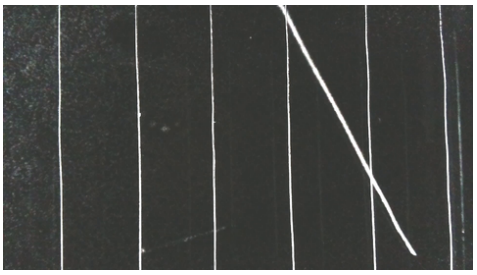


Figura 3.10: Frame 43 - esempio di traccia di una particella alfa.

La traccia viene fotografata da circa 6 frame: ciò significa che le goccioline rimangono per più di un secondo.

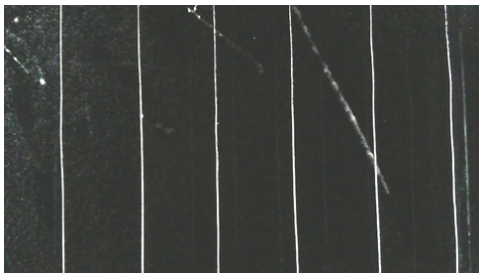


Figura 3.11: Frame 45 - esempio di traccia di una particella alfa.

Figura 3.12: Esempio di traccia di particelle alfa a fine elaborazione.

Fondo. All'interno della camera si possono trovare delle microparticelle di polvere o altri agenti indesiderati, che come detto nella Sezione 1.2 fungono da nuclei di condensazione e possono generare le gocce. Queste gocce seguono un andamento caotico, non si distribuiscono lungo un certo percorso come la traccia di una particella e vanno a formare quella nebbia di fondo che l'algoritmo di analisi delle immagini cerca il più possibile di eliminare. Alcuni accumuli di questa nebbia comunque considerati una componente con-

Figura 3.13: Esempio di fondo a fine elaborazione.

nessa dall'algoritmo. Per questo motivo un gruppo della classificazione da parte della rete saranno gli eventi di fondo.

Capitolo 4

Machine Learning

Un algoritmo di apprendimento automatico o *machine learning* è un algoritmo in grado di apprendere dai dati. Mitchell fornì questa definizione:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."[1]

Le metodologie di machine learning sono solitamente descritte nei termini di come il sistema di apprendimento processa gli esempi. Un esempio è una raccolta di caratteristiche/feature che sono state misurate quantitativamente a partire da alcuni oggetti o eventi che vogliamo che il sistema di machine learning elabori. Tipicamente un esempio viene rappresentato con un vettore $\boldsymbol{x} \in \mathbb{R}^n$, ove ciascuna componente x_i è una determinata caratteristica.

Vari tipi di problemi possono essere risolti attraverso tecniche di machine learning. Nel nostro caso, si tratta di un problema di classificazione che abbiamo affrontato con una *rete neurale convoluzionale*.
Al codice è richiesto di specificare a quale delle k categorie appartenga un certo input. Per risolvere questo task, l'algoritmo di apprendimento automatico consiste nel produrre una funzione $f : \mathbb{R}^n \to \{1, ..., k\}$. Il modello $y = f(\boldsymbol{x})$ assegna un input descritto dal vettore $\boldsymbol{x}$ ad una determinata categoria identificata dal numero y. Nel caso in esame, l'input è un'immagine - o meglio una matrice i cui valori sono la luminosità dei pixel - mentre l'output e' un codice numerico che identifica la presenza di una traccia nell'immagine. Lo scopo di questo capitolo e' quello di descrivere le componenti di una rete neurale (Sez.4.1) e gli algoritmi che permettono di apprendere dai dati (Sez.4.2). In particolare andremo a descrivere nel dettaglio le caratteristiche peculiari di una rete neurale convoluzionale (Sez.4.3).

4.1 Architettura delle Reti Neurali

Una *rete neurale artificiale*, solitamente chiamata *rete neurale*, e' un modello matematico composto da neuroni tra loro opportunamente collegati.

[1] Tom M. Mitchell (1997), *"Machine Learning"*, McGraw-Hill Science/Engineering/Math

Il più semplice esempio di neurone è il *percettrone*. Il percettrone prende un set di input binari $(x_1, .., x_n)$ e fornisce in uscita un solo output binario attraverso la funzione:

$$output = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} w_i x_i \leq threshold \\ 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq threshold \end{cases} \tag{4.1}$$

ove il valore di soglia o *threshold* e' una misura di quanto facilmente il neurone debba accendersi (cioe' fornire 1 come output), mentre i pesi $w_1, .., w_n$ sono numeri reali che esprimono la differente importanza dei singoli input rispetto all'output. E' possibile semplificare questa trattazione definendo il bias $b = -threshold$ e il prodotto scalare

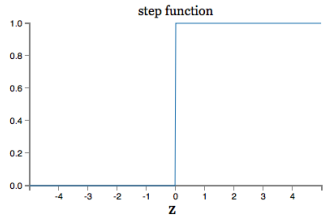


Figura 4.1: Funzione di attivazione a gradino. [5]

$$w \cdot x = \sum_{i}^{n} w_i x_i$$

ove $w = (w_1, ..., w_n)$ e $x = (x_1, ..., x_n)$ sono vettori le cui componenti sono rispettivamente i pesi e gli input. In questi termini, l'output 4.1 diventa:

$$output = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b \geq 0 \end{cases} \tag{4.2}$$

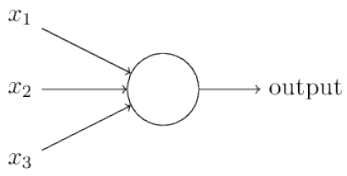


Figura 4.2: Diagramma di un percettrone [5]

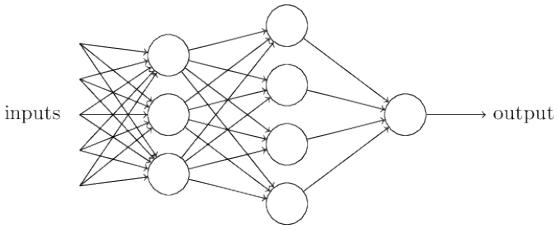


Figura 4.3: Rete neurale di un percettroni [5]

A seconda dei valori assegnati ai parametri (w, b) si ha un differente modello predittivo. Combinando vari percettroni tra loro, si possono creare strutture complesse in cui solamente alcuni di essi, detti neuroni di input, assumono in input il set di dati, mentre gli altri hanno come input l'output dei precedenti, creando delle reti di neuroni interconnessi distribuiti in vari strati o *layer*: queste sono per l'appunto le reti neurali. Il compito delgli algoritmi di machine learning è quello di trovare i pesi e i bias del modello che meglio descrive i dati, ottimizzandoli automaticamente a partire dai dati; per fare ciò, è necessario che piccoli cambiamenti dei parametri del modello conducano a loro volta a piccoli cambiamenti dell'output. Il percettrone non puo' essere l'unita' che meglio si adatta a questa necessità, in quanto il piu piccolo cambiamento del valore di output, nonche' l'unico possibile, e' 1. Per superare questa problematica dovuta alla discretezza dell'output, al percettrone e' stato sostituito un altro tipo di neurone, chiamato sigmoidale.

Il *neurone sigmoidale* prende come input un set di valori reali, che possono assumere qualsiasi

valore compreso tra 0 e 1, e ritorna un singolo output, anch'esso compreso tra 0 e 1. Definita la variabile z come

$$z = w \cdot x + b$$

che e' una combinazione pesata dei vari input, questa e' usata per valutare la *funzione logistica* o *funzione sigmoidale*

$$\sigma(z) = \frac{1}{1 + e^{-x}} \tag{4.3}$$

che e' una particolare forma di quella che viene definita in generale una *funzione di attivazione*. Dalla figura si puo' osservare come vari monotonicamente da 0 a 1; proprio questa caratteristica conferisce la proprieta' cercata: infatti il cambiamento sull'output dovuto a piccoli cambiamenti Δw_j nei pesi e Δb nei bias si puo' ben approssimare con

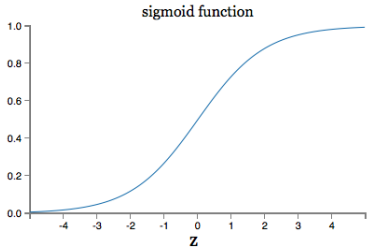


Figura 4.4: Funzione di attivazione sigmoidale. [5]

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b \tag{4.4}$$

che e' una funzione lineare di Δw_i e Δb: cio' permette, scegliendo piccoli cambiamenti nei pesi e nei bias, di ottenere un cambiamento $\Delta output$ piccolo a piacere.

Esistono vari tipologie di neuroni artificiali di utilizzo attualmente diffuso, oltre al neurone sigmoidale appena citato. La caratteristica che li accomuna è la regolarità della funzione di attivazione, proprietà necessaria come spiegato in precedenza. Un esempio delle più comuni funzioni di attivazione utilizzate nelle reti neurali artifiali sono [7]:

- **Tanh**(tangente iperbolica): $\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
  Versione riscalata della funzione sigmoidale.

- **Softmax**: $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_i}}$
  Come la funzione sigmoid costringe i valori di output ad un qualsiasi valore tra 0 e 1, ma in aggiunta li suddivide in modo che la somma totale delle uscite sia 1. L'output della funzione softmax è equivalente ad una distribuzione di probabilità tra le varie categorie, indica la probabilità che l'input appartenga ad una certa classe.

- **ReLU**(Rectified Linear Unit): $\sigma(z) = max(0, z)$
  E' la tipologia di neurone più utilizzato nelle recenti reti deep learning. Un rettificatore lineare ha come output 0 se l'input è minore di 0, mentre l'entrata stessa se questa è maggiore di 0. Il vantaggio del suo utilizzo sta nel valore della sua derivata, che è costante per valori positivi e si traduce in una maggiore rapidità di apprendimento. Una buona approssimazione di un rettificatore è la funzione analitica softplus ($f(x) = log(1 + x)$).

Come detto in precedenza, una rete neurale è una struttura complessa composta da vari strati di neuroni interconnessi. Se in linea di principio qualsiasi tipo di struttura è permessa,

le performance degli algoritmi di apprendimento dipendono fortemente dal design specifico scelto per la rete. I tipi più comuni di rete sono:

- **Feedforward neural network**: come si può vedere nell'esempio mostrato in Figura 4.5 , una rete neurale feedforward è una rete dove le connessioni tra le unità non formano cicli o feedback per cui gli outputs del modello diventano essi stessi degli input, ma le informazioni si muovono unicamente in avanti da quello che viene chiamato *strato di input* (*input layer*) allo *strato di output* (*output layer*), attraverso gli *strati nascosti* (*hidden layers*).

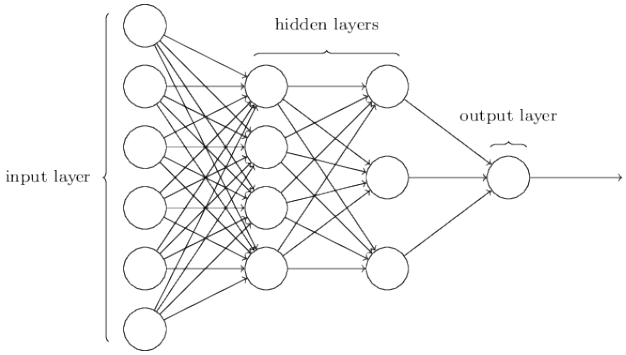


Figura 4.5: Struttura di una rete feedforward [5]

- **Recurrent neural network**: è un tipo di rete neurale in cui le connessioni formano un loop tra i vari neuroni.

Da questo punto in poi tratteremo reti di tipo feedforward.

4.2 Algoritmi di apprendimento

Nella sezione 4.1 è stato descritto il funzionamento dei neuroni e come questi si connettano a formare la rete neurale. Il compito della rete è quello di approssimare una funzione $y = f^*(\vec{x})$, definendo una mappa $y = f(\vec{x}; \theta)$ e cercando quei valori dei parametri θ affinchè si ottenga la miglior approssimazione: per fare ciò è necessario introdurre gli algoritmi che permettono questa ricerca a partire dai dati.

Le più moderne reti neurali consistono di svariati layer contenenti un gran numero di neuroni, che si traducono in un numero ancora più grande di pesi e bias. E' quindi necessario indrodurre una notazione che si userà nel seguito della trattazione. Supponendo di avere una rete con n input e L strati nascosti, m_l è il numero di neuroni del l-simo layer. Soffermandosi sul j-simo neurone del l-simo layer, questo avrà un unico bias e un numero di pesi pari al numero di output del (l-1)-simo layer. Questo significa che per ciascun neurone ci saranno $(m_{l-1}+1)$ parametri e per ciascun layer ci saranno $[m_l(m_{l-1}+1)]$ parametri. Complessivamente per gli strati nascosti ci saranno $\sum_{l=1}^{L}[m_l(m_{l-1}+1)]$.
Tutti i pesi possono essere riassunti in un tensore 3-dimensionale, dimensioni che corrispondono rispettivamente al numero del layer, al numero di neurone in quello specifico layer e all'input a cui tale peso è associato. Quindi, per il j-simo neurone nel l-simo layer i pesi sono rappresentati come $\vec{w}_j^l = (w_{j,1}^l, ..., w_{j,m_{l-1}}^l)$.

I bias invece si possono rappresentare più semplicemente con una matrice; il bias del j-simo neurone nel l-simo layer si rappresenta come b^l_j.

4.2.1 Loss function

Nel compito di ricercare questi parametri ottimali, è necessario introdurre una funzione che quantifichi il grado di apprendimento della rete; in altre parole, dato un set di valori di input $\{\vec{x}\}$ e il relativo set di output $y(\{\vec{x}\})$, detto *training sample*, tale funzione deve quantificare l'efficienza previsionale della rete, cioè quanto le previsioni del modello si avvicinano ai valor veri degli output y_{true}. Questa funzione viene chiamata *loss function* o *cost function* $C((w,b), y_{true}, y_{pred}(\vec{x}))$ e mappa gli esempi di input in un numero reale che intuitivamente rappresenta un certo valore associato ai dati. Esistono molteplici forme per la loss function, che dipende dai paramentri del modello (w,b), dai valor veri degli output y_{true} e dai valori di output previsti dalla rete $y_{pred}(\vec{x})$, ma tipicamente vengono scelte funzioni regolari per facilitare il computo del gradiente. Alcuni esempi delle più comuni sono:

- **Quadratic loss function**:

$$C(w,b) = \frac{1}{2N} \sum_{\{\vec{x}\}} \|y_{true} - y_{pred}(\vec{x})\|^2 \tag{4.5}$$

  ove N è il numero di eventi processati dalla rete, $\|\cdot\|$ la norma di un vettore n-dimensionale[2] e la sommatoria è su tutto il training sample.

- **Cross-entropy loss function**:

$$C(w,b) = \frac{1}{N} \sum_{\{\vec{x}\}} \sum_j [y^j_{true} \ln(y^j_{pred}) + (1 - y^j_{true}) \ln(1 - y^j_{pred})] \tag{4.6}$$

  ove la somma su j si intende la somma sulle n componenti dei vettori y_{true} e $y_{pred}(\vec{x})$.

Il compito della rete è quello di trovare i parametri del modello (w,b) che rendono i valori teorici il più vicino possibile ai valori previsti dalla rete: tale caratteristica è equivalente a porre la loss function $C(w,b) \approx 0$. L'allenamento di una rete neurale è quindi un problema di ottimizzazione di una certa funzione multivariata: sono necessari degli algoritmi di ottimizzazione che aggiornino i parametri per minimizzare la cost funcion il più rapidamente possibile: questi vengono chiamati proprio *optimizer*.

4.2.2 Optimizer

Come per la cost function, sono presenti in letteratura vari algoritmi di ottimizzazione. In seguito sono mostrati alcuni dei più comuni e efficienti tutt'oggi utilizzati.

Gradient descent. Rinominiamo i parametri del modello (w,b) in un unico vettore $\vec{\nu} = (\nu_1, ..., \nu_n)$ e assumiamo che sia continuo nelle sue n componenti. Data la regolarità della cost

[2] Per un vettore n-dimensionale $\vec{v} = (v_1, ..., v_n)$: $\|\vec{v}\|^2 = v_1^2 + v_2^2 + ... + v_n^2 = \sum_j v_j^2$. Si osservi che $\|\vec{v}\| \geq 0 \ \forall \vec{v}$

function C, la variazione ΔC per piccole variazioni di $\vec{\nu}$ si può esprimere come:

$$\Delta C(\vec{\nu}) \approx \frac{\partial C}{\partial \nu_1}\Delta\nu_1 + ... + \frac{\partial C}{\partial \nu_n}\Delta\nu_n \tag{4.7}$$

Definendo il vettore $\Delta\vec{\nu}$ [3], le cui componenti sono le singole variazioni $\Delta\nu_i$, e il gradiente della cost function ∇C [4], l'equazione 4.7 si può riscrivere come

$$\Delta C(\vec{\nu}) \approx \nabla C \cdot \Delta\vec{\nu} \tag{4.8}$$

Dato che l'obiettivo dell'algoritmo è quello di minimizzare la cost function, vogliamo che per piccoli aggiustamenti dei parametri questa decresca. Per fare ciò, si sceglie la variazione $\Delta\vec{\nu}$ come

$$\Delta\vec{\nu} = -\eta\nabla C \tag{4.9}$$

ove η è un parametro reale definito positivo ($\eta > 0$) che viene chiamato *learning rate* o *tasso di apprendimento*. Così facendo, dall'equazione 4.8 si ricava che $\Delta C \approx -\eta\nabla C \cdot \nabla C = -\eta\|\nabla C\|^2 \leq 0$ come cercato. Affinchè sia valida la condizione di linerità nella formula 4.7 è necessario che la variazione di ogni singolo parametro $\Delta\nu_i$ sia piccola abbastanza; questo si traduce nel vincolare la grandezza dello spostamento, fissando la norma di $\Delta\vec{\nu}$: si introduce un nuovo parametro $\epsilon > 0$ tale che $\|\vec{\nu}\| = \epsilon$. In questo modo si ottiene una "legge del moto" per i vari parametri:

$$\vec{\nu} \;\rightarrow\; \vec{\nu}\,' = \vec{\nu} - \eta\nabla C \tag{4.10}$$

che esplicitata per i pesi e i bias del generico neurone $(\vec{w}, b)^l_j$ diventa:

$$w'^l_j = w^l_j - \eta\frac{\partial C}{\partial w^l_j} \qquad b'^l = b^l - \eta\frac{\partial C}{\partial b^l} \tag{4.11}$$

che applicata ripetutamente permette di raggiungere il minimo della cost function. Ogni iterazione che consiste nel computo della cost function per il dataset di input e nell'aggiornamento dei parametri viene chiamata *epoca*.

Stochastic gradient descent. Se si osserva nuovamente l'espressione 4.5 della *funzione di costo quadratica*, ha la forma seguente:

$$C = \frac{1}{N}\sum_{\{\vec{x}\}} C_{\vec{x}} \tag{4.12}$$

che è una media delle cost function $C_{\vec{x}} = \frac{1}{2}\|y_{true} - y_{pred}(\vec{x})\|^2$ di ogni singolo evento $\vec{x}$ processato. Allo stesso modo il gradiente si può rappresentare come la media degli N gradienti

$$\nabla C = \frac{1}{N}\sum_{\{\vec{x}\}} \nabla C_{\vec{x}} \tag{4.13}$$

Da un punto di vista computazionale, calcolare per ogni iterazione la media di un numero di gradienti pari al numero di esempi del dataset è un'operazione gravosa e richiede molta

[3] $\Delta\vec{\nu} = (\Delta\nu_1, ..., \Delta\nu_n)$
[4] $\nabla C = (\frac{\partial C}{\partial \nu_1}, ..., \frac{\partial C}{\partial \nu_n})$

memoria, visto che il numero di esempi di training e il numero di parametri raggiungono anche svariati milioni o più. Per ovviare a questa problematica e velocizzare il processo di apprendimento è stato introdotto l'algoritmo di *Stochastic Gradient Descent*. L'idea è quella di stimare il gradiente ∇C calcolando $\nabla C_{\vec{x}}$ per un piccolo campione scelto casualmente tra i vari training input; mediando su questo piccolo campione, si ottiene rapidamente una buona stima di ∇C, velocizzando quindi il processo di apprendimento. Lo stochastic gradient descent lavora quindi prendendo un piccolo campione di m dati di input, che viene chiamato *mini-batch*, e stima il gradiente della funzione di costo come

$$\nabla C \approx \frac{1}{m} \sum_{m} \nabla C_{\vec{x}} \tag{4.14}$$

e aggiornando di conseguenza ad ogni epoca i pesi e i bias.
Oltre a questi due qui citati, in letteratura ne sono presenti molti altri, ma che sono versioni più sofisticate del SGD.

4.2.3 Overfitting, Underfitting e Regolarizzazione

L'obiettivo principale dell'apprendimento automatico è ottenere un algoritmo che sia performante nella classificazione di nuovi input e non solamente nella classificazione del set di esempi utilizzato durante la fase di apprendimento. L'abilità di ben performare su input non osservati durante il training è chiamata *generalizzazione*.

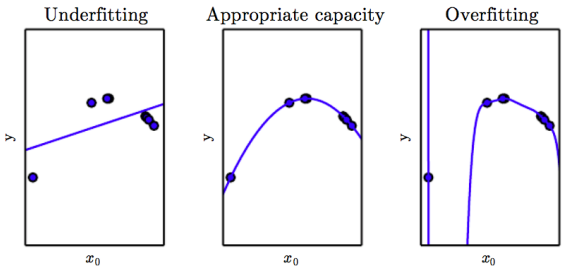


Figura 4.6: Esempio di uno stesso set di dati interpolati da un diversi modelli: quello a sinistra è affetto da underfitting, a destra invece da overfitting. [6]

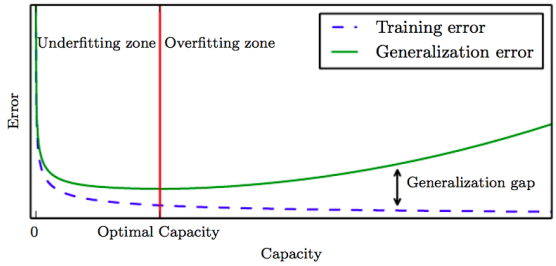


Figura 4.7: Andamento tipico degli errori di training e validazione in funzione della capacità della rete. [6]

Tipicamente, durante la fase di apprendimento, si ha accesso ad un set di training, a cui si associa una misura dell'errore del tasso di apprendimento tramite la cost function, detto training error. Ciò che distingue un algoritmo di machine learning da un semplice problema di ottimizzazione è che si cerca di minimizzare anche l'errore di generalizzazione, chiamato anche errore di test. L'errore di generalizzazione di un modello di machine learning misura la performance su un *test set* di esempi differenti rispetto al training set. I fattori che determinano quanto sia performante un algoritmo di machine learning sono le sue capacità di: ottenere un training error basso; mantenere piccola la differenza tra training e test error. A questi due fattori corrispondono le due problematiche più comuni: *overfitting* e *underfitting*. L'underfitting interviene quando il modello non è capace di ottenere un errore di training sufficientemente piccolo. L'overfitting invece quando il gap tra errore di training e di test è troppo gran-

de.
Tipicamente quella che occorre più di frequente è l'overfitting. Il gap tra training e test error può essere ridotto innanzitutto aumentando il numero di esempi con cui allenare la rete. Un'altra soluzione possibile è la riduzione della complessità della rete, diminuendo il numero di layer o neuroni. Esistono invece alcune tecniche che permettono di contrastare queste problematiche, soprattutto l'overfitting, andando a modificare l'algoritmo di apprendimento. Queste tecniche prendono il nome di regolarizzazione. Di seguito discutiamo le principali.

Regolarizzazione L2. E' la più comune forma di regolarizzazione. Detta anche *weight decay*, consiste nell'aggiunta di un termine alla loss function chiamato termine di regolarizzazione

$$\Omega(w) = \frac{\lambda}{2N}\|w\|^2 = \frac{\lambda}{2N}\sum_{l,j}(w_j^l)^2 \tag{4.15}$$

ove λ è un nuovo parametro detto *parametro di regolarizzazione*. Questo nuovo termine di conseguenza influenza anche la legge di aggiornamento dei pesi

$$w_j'^l = w_j^l - \eta\frac{\partial C}{\partial w_j^l} - \eta\frac{\lambda}{N}w_j^l \tag{4.16}$$

con un termine che li tende a rimpicciolirli. Intuitivamente, è un modo per costringere la rete a apprendere pesi di piccolo valore, fatto che riduce la complessità del modello e quindi la sua capacità di fittare il rumore dei dati.

Regolarizzazione L1. Allo stesso modo della tecnica precedente, consiste anch'essa nell'aggiunta di un termine alla cost function

$$\Omega(w) = \frac{\lambda}{N}\sum_{l,j}|w_j^l| \tag{4.17}$$

Allo stesso modo tende a rimpicciolire i pesi.

Dropout. A differenza delle regolarizzazioni L1 e L2, il dropout non va a modificare la cost function, bensì modifica la rete stessa. La tecnica del dropout consiste nell'ignorare randomicamente un certo set di neuroni degli strati nascosti durante la fase di training, cioè non vengono considerati nel computo di una certa iterazione. Ad ogni fase di addestramento, i singoli neuroni vengono mantenuti con una certa probabilità p o trascurati dalla rete con probabilità $1-p$. In questo modo si obbliga la rete ad riprodurre l'output senza il contributo di un sottoinsieme casuale di parametri. Quindi la procedura di dropout è equivalente a mediare i risultati di un gran numero di reti diverse: ciascuna rete tenderà ad avere overfitting in maniera differente e

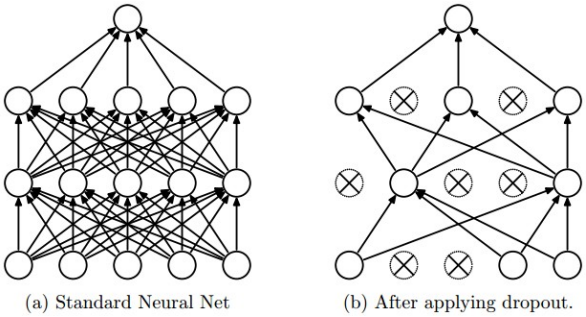

(a) Standard Neural Net (b) After applying dropout.

Figura 4.8: Esempio visivo dell'effetto che la tecnica di dropout ha sulla rete. [5]

l'effetto complessivo è quello di ridurre l'overfitting totale.

4.3 Convolutional neural network

Le *reti neurali convoluzionali* sono molti simili alle reti neurali ordinarie che abbiamo trattato nella Sezione 4.1: sono anch'esse costituite di neuroni, ciascuno con i propri pesi e bias. Ciascun neurone riceve degli input e per mezzo di un prodotto scalare e della relativa funzione di attivazione computa un valore di output. L'intera rete traduce i pixel di un'immagine da un lato ai punteggi delle classi dall'altro seguendo gli stessi algoritmi di apprendimento mostrati nella Sezione 4.2. La differenza sta nel fatto che le architetture ConvNet presuppongono esplicitamente che gli input consistano di immagini, il che consente di codificarne alcune proprietà nell'architettura e di rendere più efficiente la classificazione, riducendo notevolmente la quantità di parametri della rete.

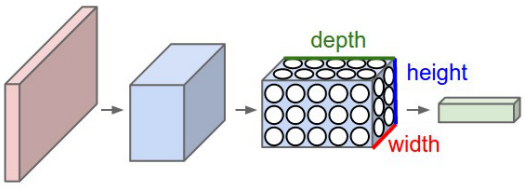


Figura 4.9: Esempio di funzionamento di un layer di una ConvNet. Ciascun layer trasforma un volume 3D di input in un volume 3D di output di neuroni di attivazione. [8]

In particolare, i layers di una ConvNet hanno neuroni disposti in 3 dimensioni: larghezza, altezza, profondità. Ad esempio. le nostre immagini di input riscalate hanno dimensione 128x128x1 (le immagini a colori RGB avranno invece 3 come dimensione finale, una per ciascuna scala di colori). Invece di avere una struttura fully-connected, i neuroni di uno strato saranno collegati solo ad una piccola regione del layer precedente (Figura 4.9). Infine, per una rete convoluzionale che classifica immagini in k classi, lo strato finale di output sarà di dimensione $1x1xk$, perchè l'architettura ridurrà l'immagine intera in un singolo vettore di punteggi delle k classi.

Come abbiamo detto in precedenza, una ConvNet è una sequenza di layer. Vengono tipicamente utilizzati tre tipi di layer per costruire un'architettura ConvNet: Layer Convoluzionale, Pooling Layer, e Fully-Connected Layer (esattamente come visto per le reti neurali standard). Andiamo quindi ad analizzare nei particolari i primi due tipi di layer.

Convolutional layer. Lo strato convoluzionale è il blocco portante di una rete convoluzionale e racchiude la maggior parte del lavoro computazionale. I parametri che lo caratterizzano sono una serie di filtri. Ciascun filtro si può vedere come un rettangolo piccolo spazialmente (in altezza e lunghezza) rispetto alle dimensioni dell'immagine; nel passaggio da un layer al successivo spostiamo il filtro su tutta l'immagine. I valori dei pixel dell'immagine vengono moltiplicati con i valori dei pixel nel filtro per mezzo di un prodotto scalare: tale operazione prende il nome di convoluzione[5]. In questo modo si viene a creare una mappa di attivazio-

[5]Dal punto di vista matematico, la convoluzione è un'operazione tra due funzioni di una variabile $x, w : \mathbb{R} \to \mathbb{R}$ ove $w(a)$ è una certa funzione di peso

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

Tale funzione può essere estesa al caso bidimensionale e a variabili discrete, usando una funzione peso K

ne bidimensionale. Intuitivamente, la rete imparerà come si attivano i filtri quando vedono alcuni tipi di caratteristiche visive come un bordo di qualche orientamento o una macchia di un certo colore sul primo strato, o eventualmente interi schemi a nido d'ape o simili a ruote su strati più alti della rete . Ora avremo un intero set di filtri in ogni livello CONV e ognuno di essi produrrà una mappa di attivazione bidimensionale separata. Raggruppando queste mappe di attivazione lungo la dimensione della profondità si produce un nuovo volume di output.

Quando si trattano input ad alta dimensione come le immagini, come abbiamo visto sopra, non è pratico collegare i neuroni di uno strato a tutti i neuroni del volume precedente. Invece, collegheremo ciascun neurone solo ad una regione locale del volume di input. L'estensione spaziale di questa connessione è un iperparametro chiamato *campo recettivo* del neurone (equivalentemente, questa è la dimensione del filtro). Questa proprietà viene definita *Connettività locale* e ha una forte analogia con i sistemi biologici. Infatti ogni voce nel volume di output 3D può anche essere interpretata come un'uscita di un neurone che guarda solo ad una piccola regione nell'input e condivide i parametri con tutti i neuroni a sinistra e a destra nello spazio.

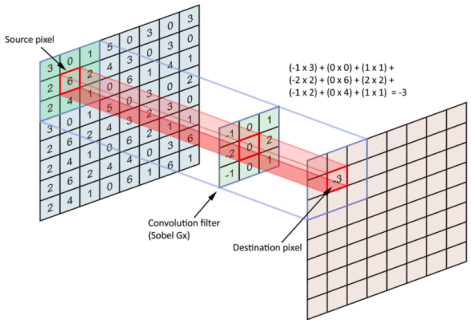


Figura 4.10: Esempio di funzionamento di un layer convoluzionale con un filtro di dimensione 3x3. [8]

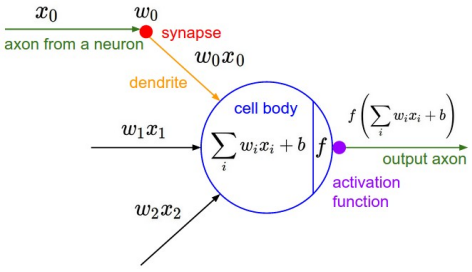


Figura 4.11: Analogia del funzionamento di un neurone artificiale di un convolutional layer e quello di un neurone animale. [8]

Le altre proprietà che caratterizzano uno strato convoluzionale sono la disposizione spaziale e la condivisione dei parametri. Tre iperparametri controllano le dimensioni del volume di uscita e la disposizione dei neuroni in esso: profondità, stride (o passo) e zero-padding. La profondità corrisponde al numero di filtri che vogliamo usare, ognuno dei quali sta imparando a cercare qualcosa di diverso nell'input. Bisogna anche specificare il passo con cui facciamo scorrere il filtro lungo tutta l'immagine. Quando il passo è 1, spostiamo i filtri un pixel alla volta, prima in altezza e poi in larghezza. Quando il passo è 2 o superiore, i filtri saltano 2 pixel alla volta mentre li facciamo scorrere. Più alto sarà il passo, più piccoli spazialmente saranno i volumi di output. A volte sarà invece conveniente riempire il volume di input con zeri tutt'attorno al bordo. La dimensione di questi zeri corrisponde all'iperparametro dello zero-padding. La caratteristica interessante dello zero-padding è che ci permetterà di controllare la dimensione spaziale dei volumi di output (più comunemente viene utilizzato per preservare esattamente le dimensioni spaziali del volume di input in modo che siano le stesse del volume di output).

bidimensionle

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Pooling layer. È comune inserire periodicamente un *pooling layer* tra i successivi strati convoluzionali in un'architettura ConvNet. La sua funzione è quella di ridurre progressivamente la dimensione spaziale della rappresentazione per diminuire la quantità di parametri: di conseguenza si riesce a ridurre la computazione nella rete e anche a controllare l'overfitting. Il Pooling Layer opera indipendentemente su ogni livello di profondità dell'input e lo ridimensiona spazialmente, sostituendo alla superifice di input un'unico output, che tipicamente è il valore massimo (*max pooling*) o il valore medio (*average pooling*) dei valori di input. La forma più comune è un raggruppamento con filtri di dimensioni 2x2 applicati con un passo di 2. In questo modo si riduce ogni livello di profondità nell'input di 2 unità sia lungo la larghezza che l'altezza, scartando il 75% delle attivazioni. La dimensione della profondità rimane invece invariata.

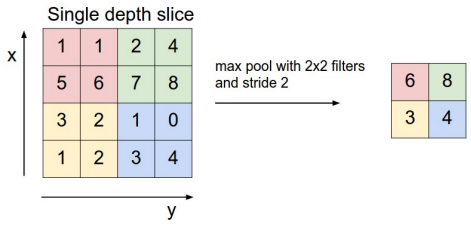


Figura 4.12: Esempio di funzionamento di un filtro di max pool di dimensione 2 e passo 2. [8]

Al termine di strati convolizionali e di pooling, solitamente si inserisce uno strato di neuroni fully-connected, preceduto da uno strato flattening. Lo strato di flattening appiattisce gli array 3D di output dei vari layer in un array monodimensionale, che è l'input per lo strato finale fully-connected.

Capitolo 5

Risultati sperimentali

L'analisi sperimentale si basa su un insieme di 3 ore di video della camera. Questi video sono stati analizzati con il codice descritto nella Sezione 3.2 per estrarre le singole tracce. Inizialmente l'obiettivo prefissato era costruire una rete che classificasse i vari tipi di particelle, alfa e beta; vista la poca statistica di particelle alfa rivelate, è stata invece creata una rete che discimina la presenza di una traccia nell'immagine[1].
Come detto nella Sezione 4.3, è stato scelto di costruire una Rete Neurale Convoluzionale.

5.1 Creazione del database

L'operazione preliminare è stata la creazione di un database contenente gli eventi estratti e classificati per poter allenare la rete neurale. Dall'analisi dei video sono state rivelate solamente qualche decina di particelle alfa, un numero troppo basso per poter creare una classe che la rete possa imparare a classificare. Per tale motivo abbiamo suddiviso le tracce estratte in due set contententi 2000 esempi ciascuno, rispettivamente con eventi e fondo. Ciascun set è stato suddiviso in due subset, uno per il training di 1600 immagini, le restanti 400 per la validazione. Uno dei limiti di questa analisi è la ristrettezza del database che ha pochi dati rispetto ai tipici database per il training di una *Deep Neural Network*.

5.2 Struttura e analisi delle reti

Struttura. La rete funziona come un classificatore binario, ossia a partire dagli input ricava una legge per distinguerli in due classi, a seconda che sia presente o meno una traccia. La cost function utilizzata è la *binary crossentropy*, un caso particolare della funzione 4.6 mostrata nella Sezione 4.2.1 in cui $j = 1$. Nella Sezione 4.3 sono stati descritti le componenti di una ConvNet. Il numero di questi layer e la loro disposizione è ciò che distingue ciascuna rete ed trovare la struttura più efficiente per questa classificazione è obiettivo dell'analisi sperimentale. Sono stati testati due pattern di layer differenti, ripetuti per certo numero di volte: un pattern consiste in una coppia di layer convoluzionali seguiti da uno strato di

[1] Il codice sorgente è disponibile al seguente link: `https://github.com/GiacomoBarzon/ConvNeuralNetwork`

pooling (Conv+Conv+Pool); un secondo costituito da una serie di n strati convoluzionali e solamente alla fine un blocco di pooling (MultiConv+Pool).

Per la creazione delle varie architetture e il loro apprendimento è stata utilizzata l'interfaccia KERAS [9], un Application Programming Interface (API) di alto livello per le reti neurali, scritto in PYTHON e in grado di funzionare su TENSORFLOW (utilizzato in questa analisi) [10], THEANO o CNTK, che sono le librerie più utilizzate per il machine learning. La libreria KERAS contiene tutti gli strumenti computazionali di base necessari per creare ed eseguire un algoritmo di apprendimento automatico basato su reti neurali.

Metodi di analisi. Durante la fase di allenamento di ciascuna architettura sono stati plottati gli andamenti di loss function e accuracy (cioè la percentuale di previsioni corrette), sia per il set di training che per quello di validazione, per monitorare la stabilità del training e la presenza di un eventuale over-fitting[2].

Per valutare e comparare tra di loro i vari modelli, è stato utilizzato il valore di *Area Under the ROC Curve* (AUC). ROC è un acronimo per *Receiver Operating Characteristic* e attraverso l'analisi di queste curve è possibile valutare la capacità di un classificatore binario di discernere. In questo caso la curva ROC è stata valutata come sensibilità in funzione della (1-specificità) al variare del punto di soglia. In altre parole, corrisponde a valutare la frazione di veri positivi (nel caso in esame, eventi contenti una traccia che la rete predice avere output maggiore del valore di soglia) in funzione dei falsi positivi (eventi di fondo che la rete predice avere output maggiore del valore di soglia). Infatti la rete predice con una certa probabilità la presenza o meno di una traccia: nel caso dell'accuracy, tale soglia è posta a 0.5 e per valori superiori la rete predice la presenza della traccia. Invece nel caso della curva ROC viene stimata la precisione delle predizioni al variare del valore di soglia, che assume valori tra 0 e 1. Il valore di AUC invece può variare tra 0.5 (classificatore casuale) e 1 (classificatore perfetto). Una modello predittivo è tanto più efficiente quanto il valore di AUC è vicino all'unità.

5.3 Setup degli iperparametri

Dopo aver fatto dei test preliminari, i seguenti iperparametri vengono mantenuti costanti per tutte le architetture.

Funzioni di attivazione. Per il layer di output è stata scelta la funzione *sigmoid*. Tale funzione può essere paragonata ad una densità di probabilità tra i valori 0 e 1. Assumendo che il valore 1 corrisponda ad un esempio contenente una traccia mentro il valore 0 da un evento di fondo, l'output della funzione sigmoid si può interpretare con la probabilità che in un determinato esempio sia presente una traccia. Per l'input e hidden layers è stata scelta invece la funzione *ReLU*. Questa scelta permette di velocizzare il training della rete: infatti la derivata è costante, quindi semplifica il costo computazionale, e non tende a saturare verso 0 per valori di input grandi come invece accade per la funzione sigmoidale.

Optimizer. E' stato scelto l'algoritmo Adam (Adaptive Moment Estimation), che è un'estensione del SGD descritto in Sezione 4.2.2. A differenza del SGD, l'algoritmo Adam utilizza

[2] I vari plot si possono vedere al seguente link: `https://drive.google.com/drive/folders/1Zuz2K5E2h4Bmc_MSok7UrgDeHbHcIXXO?usp=sharing`

un tasso di apprendimento adattivo: adatta il tasso di apprendimento ai parametri, eseguendo aggiornamenti più piccoli (cioè bassi tassi di apprendimento) per i parametri associati a caratteristiche ricorrenti e aggiornamenti più grandi (cioè alti tassi di apprendimento) per parametri associati a caratteristiche poco frequenti[3].

Mini-batch size. La grandezza del *mini-batch* è stata posta a 64. Per valori più bassi si è osservato che l'andamento della cost function è rumorosa e oscilla, sintomo che indica che la dimensione del batch potrebbe essere troppo piccola.

Learning rate. Il monitoraggio della cost function è utile anche per valutare il tasso di apprendimento. Per identificare il valore ottimale, si è allenata la rete con un valore di learning rate sovrastimato e con valori via via più piccoli, fino a che la cost function ha assunto un andamento il più possibile decrescente monotonicamente, come mostra la Figura 5.3. Tale valore ottimale è stato individuato in $\eta_0 = 0,0001$.

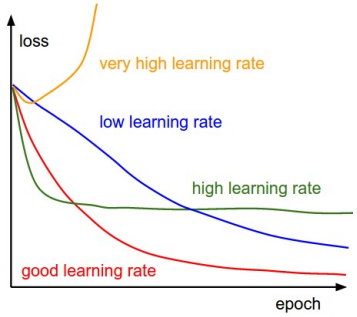


Figura 5.1: Esempio di andamento di cost function per learning rates differenti. [8]

Numero di epoche. Il numero di epoche è stato fissato a 40, perchè si è osservato che l'andamento di training e validation error tende a saturare ad un valore stabile. Allenando la rete per ulteriori epoche si andrebbe incontro ad overfitting.

Dropout. E' stato posto dopo lo strato fully-connected con un valore di $0,25$.

Image augmentation. Gli esempi con cui si allena la rete grosso modo hanno lo stesso riquadro di delimitazione di pixel neri, la stessa larghezza e sono ben orientate. Se si considera ora cosa succede quando si presenta alla rete una versione leggermente spostata, ridimensionata e ruotata di un'immagine da testare, la sua capacità di prevedere la classe corretta è destinata a calare. Idealmente vorremmo insegnare al modello di essere invariante rispetto a tali distorsioni. Il modello può però solo imparare dai campioni che gli sono stati forniti. Per ovviare a questa problematica si aumenta artificialmente il numero dei dati applicando delle distorsioni alle immagini: in questo modo si istruisce la rete a meglio comportarsi in presenza di nuove tracce.

Sono stati applicati vari filtri per l'*image augmentation*: rotazione, traslazione orizzontale e verticale, ingrandimento, flip orizzontale e verticale, shear mapping (che distorge l'immagine spostando ogni punto in direzione fissa).

5.4 Risultati

Conv+Conv+Pool. Sono state allenate delle architetture con 2,3,4 blocchi in successione. Altri parametri che sono stati variati sono il numero di filtri degli strati convoluzionali (32 e 64) e il size dei pooling layer (2 e 3). In Tabella 5.1 sono mostrati i risultati per le reti con 32 filtri negli strati convoluzionali, mentre in Tabella 5.2 i risultati delle reti con 64 filtri.

[3]Per una descrizione più precisa sui vari algoritmi di ottimizzazione si veda: Sebastian Ruder, *An overview of gradient descent optimization algorithms*, `https://arxiv.org/abs/1609.04747`

$N°$ blocchi	Pooling Size	$N°$ parametri	Accuracy	AUC
2	2	13.808.033	0,951	0,982
2	3	2.388.385	0,959	0,984
3	2	2.406.881	0,959	0,987
3	3	113.121	0,945	0,980
4	2	328.225	0,955	0,988

Tabella 5.1: Performance della rete con blocchi ConvConvPool per ConvLayer a 32 filtri.

$N°$ blocchi	Pooling Size	$N°$ parametri	Accuracy	AUC
2	2	27.670.337	0,951	0,984
2	3	4.831.041	0,945	0,983
3	2	4.904.897	0,970	0,990
3	3	317.377	0,945	0,982

Tabella 5.2: Performance della rete con blocchi ConvConvPool per ConvLayer a 64 filtri.

Si può osservare una generale indipendenza delle performance delle reti dal numero di parametri di cui è costituita. Non è quindi tanto importante il numero di paramentri, bensì come questi sono distribuiti nei vari strati e come interagiscono tra di loro. Si può osservare invece un aumento delle performance della rete utilizzando un pooling size più piccolo. L'utilizzo di un pooling size più alto probabilmente porta ad una perdita di dettagli che consegue in una perdita di precisione nella classificazione. Per le reti con pooling size uguale a 2 si può notare anche un andamento crescente all'aumentare del numero di blocchi: questo significa che una rete più complessa riesce a apprendere più finemente le caratteristiche delle immagini.

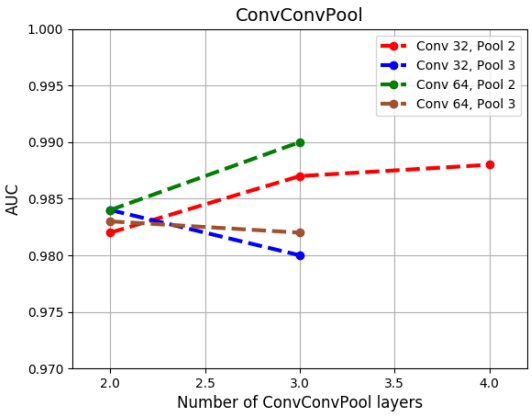


Figura 5.2: Andamento del valore di AUC in funzione del numero di layer per blocchi ConvConvPool.

MultiConv+Pool. Sono state allenate delle reti con un numero di blocchi variabile da 2 a 8 e uno strato di pooling finale, mentre da 2 a 7 con due strati di pooling finali, con pool size 2 e 4. Il numero di filtri dei ConvLayer è stato fissato a 32. In Tabella 5.3 sono mostrati i risultati per le reti con un solo layer di pooling, mentre in Tabella 5.4 i risultati delle reti con 2 pooling.

$N°$ blocchi	Pooling Size	$N°$ parametri	Accuracy	AUC
3	2	60.984.705	0,916	0,974
3	4	14.765.441	0,901	0,960
4	2	59.011.489	0,911	0,973
4	4	14.774.689	0,941	0,980
5	2	57.071.041	0,953	0,985
5	4	13.817.281	0,951	0,978
6	2	55.163.361	0,944	0,985
6	4	13.826.529	0,954	0,986
7	2	53.288.449	0,960	0,985
7	4	12.901.889	0,958	0,987
8	2	51.446.305	0,945	0,985
8	4	12.911.137	0,954	0,987

Tabella 5.3: Performance della rete con blocchi MultiConv con un Pooling layer.

$N°$ blocchi	Pooling Size	$N°$ parametri	Accuracy	AUC
3	2	14.765.441	0,897	0,959
3	4	822,657	0,915	0,971
4	2	14.774.689	0,941	0,974
4	4	831.905	0,933	0,976
5	2	13.817.281	0,939	0,975
5	4	841.153	0,935	0,975
6	2	13.826.529	0,955	0,982
6	4	850.401	0,921	0,977
7	2	12.901.889	0,965	0,989
7	4	859.649	0,925	0,981
8	2	12.911.137	0,966	0,989
8	4	868.897	0,926	0,978

Tabella 5.4: Performance della rete con blocchi MultiConv con due Pooling layer.

Si può osservare come ci sia un miglioramento dell'efficienza della rete all'aumentare del numero di layer convoluzionali. Dall'aggiunta del sesto layer in poi, il valore di AUC non aumenta più: questo può significare che la rete ha raggiunto l'efficienza massima nell'apprendere le caratteristiche delle immagini contenute nel database. Inoltre si può vedere come si abbia un'efficienza maggiore per un valore totale di pool size pari a 4 (1 pooling layer con size 4 oppure 2 pooling layer con size 2). Per un valore di pool size si ha quindi una restrizione troppo

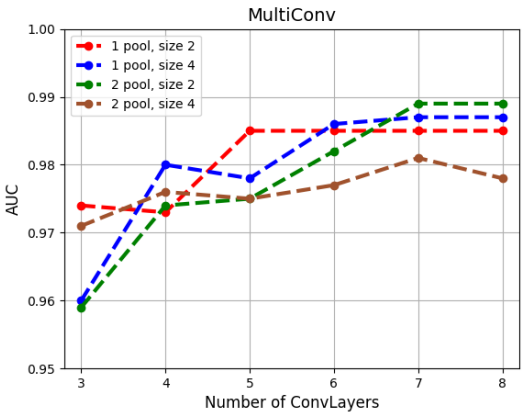


Figura 5.3: Andamento del valore di AUC in funzione del numero di layer per blocchi ConvConvPool.

grossolana dell'immagine, mentre per un pool size di 8 un restrizione troppo fine che porta alla perdita di dettagli dell'immagine.

Per tutti le varie architetture si sono ottenuti comunque ottimi valori di accuracy ($> 90\%$) e AUC ($> 95\%$). Tale risultato è una prova della bontà degli iperparametri scelti, che sono stati mantenuti costanti per tutte le varie architetture. Se si prendono in considerazione gli andamenti di accuracy, si osserva che il valore di validation accuracy è maggiore di quello di training. Ciò significa che il modello riesce a generalizzare bene. Questo deriva dal fatto che il modello è stato addestrato con dati che hanno subito augmentation, mentre il valore di validation accuracy è calcolato su un set non distorto, che è più semplice da classificare.

Capitolo 6

Prospettive future

Come detto in precedenza, il limite principale di questa trattazione è la limitatezza del numero di esempi con cui si è allenata la rete. Tale problematica può essere superata acquisendo più ore di video della camera. In questo modo si potrà ottimizzare la rete convoluzionale e migliorare la capacità di discriminare la presenza di tracce nelle immagini.
Inoltre, acquisendo più eventi si potrà convertire questa rete da classificatore binario a classificatore multiclasse, per poter catalogare i vari tipi di particelle (beta, alfa, magari anche muoni). Per fare ciò, bisognerà modificare alcune caratteristiche della rete, ad esempio non usare la funione di attivazione sigmoid ma la funzione softmax, utilizzare la funzione di costo categorical-crossentropy e non quella binaria.
Infine si potranno applicare algoritmi di apprendimento non-supervisionato, come la clusterizzazione, per studiare la classificazione delle tracce senza bias esterni provenienti dall'esperienza umana.

Bibliografia

[1] C.T.R. Wilson (1911)
On a Method of Making Visible the Paths of Ionising Particles through a Gas
Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 85, 285–288

[2] A. Langsdorf (1938)
A Continuously Sensitive diffusion Cloud Chamber
Review of Scientific Instruments 10, 91–103 (1938)

[3] I. Esteban Muñoz (2015)
Detection of particles with a cloud chamber
Universidad del Pais Basco
`https://addi.ehu.es/handle/10810/17990`

[4] F.Tortorella
Teoria e Tecniche di Interpretazione delle Immagini
Corso di Laurea in Ingegneria delle Telecomunicazioni, Facoltà di Ingegneria
Università degli Studi di Cassino
`http://webuser.unicas.it/tortorella/TTII/slides2004.html`

[5] Michael A. Nielsen (2015)
Neural Networks and Deep Learning
Determination Press
`http://neuralnetworksanddeeplearning.com/index.html`

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016)
Deep Learning
An MIT Press Book
`http://www.deeplearningbook.org`

[7] ReLU and Softmax Activation Functions
`https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions`

[8] CS231n Convolutional Neural Networks for Visual Recognition
`http://cs231n.github.io`

[9] Keras Documentation
`https://keras.io`

[10] TensorFlow Documentation
`https://www.tensorflow.org`